

Creare un Crack Me Linux utilizzando le Qt Libraries – Le basi delle basi delle Qt Libraries

Publicato da **bIAAd!** il 09/02/2001

Livello **base**

Introduzione

Linux è notoriamente un sistema 'free' & 'open source', quindi la stragrande maggioranza dei programmi che girano sotto questo sistema operativo, non necessitano di particolari attenzioni da parte del 'cracker'. Nulla però ci impedisce di creare degli appositi 'Crack Me' in stile Window\$.

Quello che cercherò di fare in questo tute, è di spiegare come realizzare un Crack Me con vari 'push button' e 'message box' varie, utilizzando le comuni Qt libraries (è richiesta in questo caso una minima conoscenza del C++, in quanto le Qt sono progettate in una ottica ad 'oggetti'. Questo vi assicuro, facilita di molto il lavoro....).

Le librerie le troverete molto probabilmente, già installate sul vostro Linux, ed ancor più probabilmente sotto la directory /usr/lib/qt***/. Un consiglio che vi posso dare, è di andarvi a leggere i vari tutez presenti (anche perchè molto di ciò che troverete qui, è preso da lì :).

Un'ultima cosa sulle librerie Qt: sembra che si sia da poco risolta la questione sulle licenze GPL (cioè un tipo di licenza creata dalla GNU, che lega il programma alla filosofia 'free').

Prima la sua licenza era di tipo QPL (con maggiori restrinzioni rispetto alla GPL). Le versioni precedenti la 2.2 quindi (compresa la nostra 1.44) presentano dei lievi 'handicap' a livello di documentazione e di sorgente (comunque niente di grave).

FINE DELL'ANSA.

Scarica l'allegato.

Programmi usati

- **g++**: compilatore c++ (GNU)
- **librerie Qt**: librerie grafiche

Iniziamo

Le librerie Qt forniscono all'interno di Linux, l'ossatura su cui si basa il sistema KDE. Nascono da una idea di un norvegese, Troll (un troll??), e presentano oltre ai tradizionali componenti visuali per la creazione di finestre, edit line, message box, ecc., anche un alquanto semplificato sistema di gestione degli eventi. Ma andiamo con ordine, analizzando gli elementi base che riguardano l'utilizzo delle Qt.

Ho incluso nel tutorial il file di -making- e di codice, relativi al programma finale. Per gli altri esempi più brevi, controllate quelli allegati alla documentazione delle Qt libraries (ho comunque inserito i listati)...

Consideriamo il classico programma HELLO WORLD (lo traggio uguale, dai tutorials ufficiali delle librerie Qt):

```
#include <qapplication.h>
#include <qpushbutton.h>

int main (int argc, char **argv)
{
    QApplication app(argc, argv);
    QPushButton hello("Hello World");

    hello.resize(100,30);
    app.setMainWidget (&hello);
    hello.show();
    return app.exec();
}
```

Questo breve programma, include dei primi importanti elementi:

La classe QApplication, ci è fornita da "qapplication.h", ed eredita public, QObject. QObject è l'elemento base per le Qt libraries (un po' come l'Object per le classi Java;-). Per ogni nostra applicazione che sfrutti queste librerie, deve essere sempre istanziata la classe QApplication (nell'esempio sopra, il nostro oggetto istanziato prende il nome di app).

QPushButton, come è facile intuire, rappresenta la classe relativa ad un bottone standard, che appare sulla nostra finestra. Questo elemento fa parte di una categoria più generale di elementi, i così detti widgets (che ereditano dalla classe QWidget). Questi ultimi non sono altro che "user interface object", cioè elementi grafici a disposizione dell'utente, come push buttons, labels, edit lines, ecc.. Analizziamo brevemente il semplice programma sopra:

I primi 2 include importano all'interno del nostro codice, le classi QApplication e QPushButton usate nel programma (magari date uno sguardo ai file *.h). Viene quindi creato l'object app da QApplication, ed il pulsante hello. Quest'ultimo viene ridimensionato tramite la funzione resize (ampiezza, altezza) ed avrà scritto sopra 'Hello UIC'. resize è solo una delle molte funzioni presenti in

QPushButton; per questo, un mio consiglio è di consultare il più possibile le specifiche fornite dalla TrollTech per le varie componenti, in quanto possono risultare presenti molte opzioni interessanti.

L'istruzione `app.setMainWidget(&hello);` pone il nostro pulsante come widget principale per la nostra applicazione. Se esso ad esempio viene terminato, tutta la nostra applicazione termina.

L'istruzione `hello.show();` come è intuitivo, visualizza il nostro pulsante. Questa funzione deve essere sempre richiamata dopo la creazione di un widget, affinché venga visualizzato.

Infine c'è `return app.exec();`. Qui la funzione `main()` passa il controllo a quest'ultima funzione, e non ritornerà sino alla conclusione della applicazione.

Ma come abbiamo detto, il main widget è il nostro bottone, cosa che in un interfaccia più complessa potrebbe risultare assurdo. In effetti le librerie Qt permettono di creare una sorta di gerarchia degli elementi, istanziando ad esempio un widget mother ed un widget child. Mostriamo questo concetto riscrivendo il codice dell'esempio sopra:

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qfont.h>

int main (int argc, char **argv)
{
    QApplication a (argc,argv);
    QWidget w;
    w.resize(200,120);
    QPushButton hello("Hello UIC", &w);
    hello.move (62,40);
    hello.resize (75,30);
    hello.setFont(QFont ("Times",18,QFont::Bold));
    QObject::connect(&hello,SIGNAL(clicked()),&a,SLOT(quit()));
    a.setMainWidget(&w);
    w.show();
    return a.exec();
}
```

Qui la cosa si fa più interessante. Vediamo perchè:

Innanzitutto c'è la creazione dell'oggetto `w` dalla classe `QWidget`. `w` è un widget mother, cioè un widget a cui possono essere legati dei widget minori (detti child). Questi 'minori' possono essere degli edit line, push button, ecc.. Nell'istruzione seguente viene dimensionato il nostro widget mother tramite i valori `width=200` ed `height=120` (in poche parole viene dimensionata la nostra area di 'disegno' dell'interfaccia) attraverso l'istruzione 'resize'.

Segue poi la creazione del bottone 'quit' attraverso `QPushButton("Quit",&w);`. La referenza `&w` sta ad indicare, che il nostro bottone 'poggia' sul widget mother `w` (riguardate invece l'esempio precedente). Ora `QPushButton` è un widget child. Seguono poi il suo posizionamento tramite `move (x,y)` ed il suo ridimensionamento tramite `resize`. Inoltre viene impostata la font "Times", 18 per dimensione del carattere e `QFont::Bold` per il...bold(notate il riferimento diretto a costante ::).

Ora viene il bello. Osservate l'istruzione:

```
QObject::connect(&hello,SIGNAL(clicked()),&a,SLOT(quit()));
```

Questa istruzione fa in modo tale che, ogni volta che venga premuto il bottone che appare sulla finestra, il programma termini.

`QObject::connect()` infatti (funzione membro della classe `QObject`), gestisce gli eventi delle GUI, che vengono definiti e riconosciuti come dei SIGNAL/SLOT. Entrambi non sono altro, all'interno di un oggetto, che delle semplici funzioni, che permettono la comunicazione tra gli oggetti.

I segnali possono essere –emessi– da una classe, e da tutte le relative sottoclassi. Questi vengono emessi quando lo stato interno di un oggetto cambia. Se ad esempio andate ad osservare le specifiche delle qt libraries relative al `QLineEdit`, troverete una funzione legata alla pressione del tasto return (questo ci tornerà utile più avanti), che emetterà un segnale proprio al verificarsi di tale evento. Le funzioni slot sono soltanto void, sono generati dal moc (ne parleremo più avanti), ed inoltre sono completamente indipendenti da qualsiasi GUI loop (asincroni). Infine, se più slots sono associate ad uno stesso segnale, esse verranno eseguite in maniera sequenziale ma in modo non ordinato. Questo, grosso modo, è tutto quello che ci serve sapere sui segnali.

Le slots sono delle normali funzioni all'interno di una classe, che vengono attivate quando un particolare segnale ad esse connesso, viene emesso. Si possono filtrare gli accessi a tali funzioni usando i classici `public private protected`, dove i primi le rendono accessibili anche a segnali di classi esterne, i secondi a segnali interni alla classe stessa, escludendo le sottoclassi, e gli ultimi per segnali interni alla classe e alle relative sottoclassi.

Ora siamo in grado di capire meglio l'istruzione precedentemente evidenziata:

`connect` ha come argomenti (la funzione è overloaded, quindi il numero e/o tipi di argomenti possono variare. Qui mi riferisco al caso specifico e farò così, anche per altre funzioni) un puntatore al nostro pulsante tramite `&hello`, che emette un segnale quando viene premuto, `clicked()`, ed è connesso alla nostra `QApplication a`, tramite `&a`, che termina, chiudendo la finestra, tramite la funzione `quit()`.

Cosa succede però, se creando una nostra interfaccia, non abbiamo una gestione degli eventi adeguata al nostro caso specifico?

Niente di più semplice. È infatti possibile editare in una nostra classe widget: `QWidget`, dei SIGNALS e/o SLOTS, includendo la macro `Q_OBJECT`, e permettendo il dialogo all'interno della classe stessa (e non solo) tra i vari widgets. Si può infatti estendere o

creare da zero, un widget che può includere a sua volta molti altri widgets. All'interno della classe poi, è possibile creare delle funzioni SIGNAL e/o SLOT, che permettano il dialogo tra questi. Ma vediamo un esempio che includa tutto:

```
#include <qapplication.h>
#include <qfont.h>
#include <qpushbutton.h>
#include <qwidget.h>

class Controllo : public QWidget {
    Q_OBJECT
public:
    Controllo(QWidget *parent=0, const char *name=0);

public slots:
    void esci();

private:
    QPushButton* quit;
};

Controllo::Controllo (QWidget *parent, const char *name)
:QWidget (parent,name)
{
    quit = new QPushButton ( "Quit",this, "quit");
    quit->move( 47, 40 );
    quit->resize( 106, 30 );
    quit->setFont( QFont( "helvetica", 18, QFont::Bold ) );

    connect( quit, SIGNAL(clicked()), SLOT(esci()));
}

void Controllo::esci() { quit() }
```

Questa classe, con relativo costruttore e funzione slot, all'interno di un programma fornisce lo stesso risultato finale dell'esempio precedente, ma tecnicamente porta con sè profonde differenze. Innanzitutto la classe Controllo eredita public la classe QWidget (che a sua volta eredita da QObject). Subito dopo la parentesi '{' troviamo la macro Q_OBJECT, che permette di creare all'interno della nostra classe, i SIGNAL/SLOT necessari, tramite public slots e signals. Ora nel costruttore poniamo, oltre all'istanziamento del pulsante, la sua gestione tramite la funzione connect(), che in questo caso richiede solo un puntatore (quit), in quanto la SLOT a cui è connessa, è interna alla classe stessa. Altra cosa molto importante da notare, è come all'interno dello header del costruttore, gli argomenti siano posti a 0. In questo caso avremo una top level window (abbiamo ridefinito gli argomenti nel costruttore, come quelli della classe QWidget da cui eredita). const char *name stà ad indicare l'ID della finestra (e non la title bar), mentre parent, come dicevamo, impostato a 0 dà una posizione di gerarchia "superiore" al nostro widget.

Implementando il tutto tramite la funzione main() seguente, avremo la nostra finestra con tanto di pulsante centrale che la chiude:

```
int main (int argc,char **argv)
{
    QApplication a (argc,argv);
    Controllo w;
    a.setMainWidget( &w );
    w.show();
    return a.exec();
}
```

Provate ora a compilare il programma.... Probabilmente (anche se tutto è stato fatto correttamente), avrete un errore in output, relativo alla mancanza di una table pointer, cioè una tabella con dei puntatori alle funzioni di slot/signal, che si riferisce nel nostro caso, alla sola funzione esci(). Come possiamo risolvere il problema?? Qui entra in gioco il moc (acronimo di META OBJECT COMPILER). Il moc legge le dichiarazioni delle classi, e genera un file sorgente C++ che contiene tutti i nomi degli slots e dei signals interni alla classe. Questo file deve essere poi incluso nel nostro codice, per venir compilato e linkato con il programma (guardate il programma finale, l'include sopra main()).

Il moc meriterebbe un discorso più approfondito, ed in questo tute ci limitiamo a quanto detto.

Ora abbiamo tutti gli elementi per creare il nostro crackme.

Ho allegato al tutorial il file che viene compilato dal moc, e la cui estensione è .moc. Questo file viene modificato automaticamente in fase di compilazione, quando viene aggiunta ad esempio una nuova slot od un signal nella classe Controllo del nostro programma sotto. Ho allegato anche il Makefile che ho prelevato da degli esempi inclusi nei tutorials, e modificato in base al nostro caso. Il programma sotto dovreste chiamarlo crackme.cpp (o come volete, a patto di cambiare il nome nei

makefiles). Può accadere che in fase di compilazione, all'interno del file .moc, ci sia un'include con percorso non esteso, e che genera un errore. In questo caso semplicemente, editate il file, estendete il percorso, salvate, uscite e ridate make. Potete anche modificare il Makefile, ma l'ho lasciato invariato, in quanto il path delle librerie qt, può variare:). Il path l'ho esteso nel programma sotto (ho messo il mio):

```
#include "/usr/include/qt/qapplication.h" // i paths degli header files installati sul mio
sistema
#include "/usr/include/qt/qpushbutton.h"
#include "/usr/include/qt/qfont.h"
#include "/usr/include/qt/qlinedit.h"
#include "/usr/include/qt/qlabel.h"
#include "/usr/include/qt/qwidget.h"
#include "/usr/include/qt/qmessagebox.h"

#include "string.h"

class Controllo : public QWidget {
    Q_OBJECT // macro necessaria se si vogliono immettere slots e/o signals
public:
    Controllo(QWidget *parent=0, const char *name=0);
    // costruttore che imposta il nostro widget a top level window

public slots:
    void decodifica(); // la nostra slot

private:
    QPushButton* quit; // definizione dei widgets child che useremo nel nostro crackme
    QLabel *qlab; // chiaramente potete inserirne a vostro piacimento
    QLabel *info;
    QLineEdit *qle;
};

// codice del costruttore della classe Controllo
Controllo::Controllo (QWidget *parent, const char *name):QWidget (parent,name)
{
    setCaption ("title bar"); // immette il titolo alla finestra
    setMinimumSize( 200, 180 );
    // misure massime e minime per la finestra coincidono
    setMaximumSize( 200, 180 ); // quindi non è possibile far variare la sua misura

    quit = new QPushButton ( "Check It",this,"check" ); // crea il pulsante che useremo per il
controllo del seriale
    quit->move( 47, 40 );
    quit->resize( 106, 30 );
    quit->setFont( QFont( "helvetica", 18, QFont::Bold ) );
    qlab = new QLabel("SERIALE : ",this,"hl"); // crea l'etichetta da affiancare alla edit line
per il seriale
    info = new QLabel("",this,"nfo"); // nuova etichetta "espansa" come Box in cui scrivere
informazioni personali e regole per il crack me
    info->setFrameStyle( QFrame::Box | QFrame::Sunken);
    info->setText ("info e condizioni di regolamento"); // controllare le specifiche per le
informazioni varie di dimensionamento, etc...
    info->move(1,75);
    info->resize(200,100);
    info->setAlignment(AlignHCenter | AlignVCenter);
    qle = new QLineEdit(this); // creazione della edit line su cui viene inserito il seriale per
un massimo di 15 caratteri
    qle->setMaxLength(15);
    qle->move(80,1);
    qle->resize(120,30);

    connect( quit, SIGNAL(clicked()), SLOT(decodifica())); // 2 connects che controllano il
seriale quando viene premuto il pulsante
    connect( qle, SIGNAL(returnPressed()), SLOT(decodifica())); // o quando viene premuto enter
nella edit line
}

// codice della funzione decodifica() interna allla classe Controllo
void Controllo::decodifica()
{
    bool flagga=FALSE;
    // dove inserirete la vostra eventuale routine di crypt del codice
}
```

```

// gle->text() preleva il seriale dalla edit line
if (flagga == TRUE) // condizione che si verifica nel caso si indovini il seriale
QMessageBox::information ( 0,"titolo della message box","INDOVINATO", QMessageBox::Ok);
else
QMessageBox::information ( 0,"titolo della message box","SBAGLIATO", QMessageBox::Ok);
// le message box sono varie nelle qt, quindi vi consiglio di controllare le specifiche
// chiaramente questo è solo una semplice impostazione di routine di crypt e controllo
%che
// potete variare a vostro piacimento
}

#include "crackme.moc" // importante : imposta la tavola di puntatori per le slots e i
signals

int main (int argc, char **argv)
{
QApplication a (argc,argv);
Controllo w; // questa riga sostituisce ora QWidget w;
a.setMainWidget( &w );
w.show();
return a.exec();
}

```

Cosa fa il programma?? Rileva la pressione del pulsante, o del tasto enter all'interno della edit line. E visualizza la message box relativa al successo o meno. Chiaramente dovrete inserire una routine di protezione vostra. Il programma, una volta capito il funzionamento generale delle librerie Qt, può essere tranquillamente stravolto creando un crackme ad esempio, del tipo nome/seriale, e così via.

Più generalmente abbiamo:

- 1) Istanza della applicazione
- 2) Creazione di un widget che includa un pulsante ed una edit line
- 3) Una slot che rilevi la pressione del pulsante, e controlli il seriale inserito
- 4) L'inclusione del file .moc
- 5) il passaggio del controllo dalla funzione main() alla applicazione, tramite la funzione exec()

Questo è quanto.

c ya folx!!

-eLeNg-

Conclusioni

Un saluto a tutti quelli della RACL ovviamente.

Sicuramente questo tutorial non è esauriente riguardo all'argomento delle Qt libraries, e tantomeno pretende di esserlo.

Comunque ho cercato di fondere un po' delle informazioni dei tutorials ufficiali con la mia esperienza personale, cercando di rendere estremamente facile il discorso (che poi non è particolarmente complesso), sperando che il tutto serva almeno ad avvicinare qualcuno alla programmazione ed al reversing sotto Linux (il che sarebbe veramente molto), magari creando e risolvendo crackme (per iniziare vè più che bene). Sono molto graditi degli approfondimenti o delle correzioni sull'argomento. Ok, questo è quanto.) I A O ;-

Attenzione

Questo tutorial è a solo scopo informativo. Ne io, ne i possessori del sito, siamo responsabili di un eventuale abuso di ciò che è stato trattato, per scopi di pirateria e violazione di copyright.

In qualsiasi caso rubare è sbagliato, e i vari tutorial presenti sul sito dovrebbero aiutare a comprendere l'impegno profuso da parte dei programmatori, per la creazione d'ogni singolo programma.