How to ignore ssl errors to get https website work on QML Webview

This article explains how to ignore SSL errors to get https website's work on QML Webview

Exact Problem

Many of our day to day application development requires us to use QML Webview and open some https website for transferring some secured content, in some most of the Symbian^3 devices, there's a problem with this activity.

Most of the times QML Webview fires onLoadFailed signal whenever we try to open any of the https based website. For example this link https://identi.ca/api/oauth/authorize is used during implementing OAuth to access the services of identi.ca, but everytime you try to open this url in QML Webview it will always give you an onLoadFailed error.

Exact Reason

The main reason behind the behavior mentioned above is the missing or invalid certificate on the symbian^3 device which causes SSL error, hence QML Webview fires the onLoadFailed signal.

Solution

This code snippet will explain how to tackle the situation explained above. The problem is solved by simply bypassing the warnings which are caused due to the missing SSL certificate. This can be done by providing our own custom QNetworkAccessManager to the QML Declarative Engine which ignores the SSL errors.

Code Snippet

Following class explains how we can create our own QNetworkAccessManager, and how we can make the QDeclarativeEngine to use that QNetworkAccessManager. Apart from this our own QNetworkAccessManager also takes care of ignoring the SSL errors.

customnetworkmanagerfactory.h

```
#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QSslError>

// need to derive your class from QDeclarativeNetworkAccessManagerFactory, Our class is
derived also from QObject
// the reason is to use signal and slot mechanism
class CustomNetworkManagerFactory : public QObject,public
QDeclarativeNetworkAccessManagerFactory
{
    Q_OBJECT
public:
    explicit CustomNetworkManagerFactory(QObject *parent = 0);
    virtual QNetworkAccessManager *create(QObject *parent);

public slots:
    void onIgnoreSSLErrors(QNetworkReply* reply,QList<QSslError> error);

private:
    QNetworkAccessManager* m_networkManager;

};
```

customnetworkmanagerfactory.cpp

```
#include "customnetworkmanagerfactory.h"

CustomNetworkManagerFactory::CustomNetworkManagerFactory(QObject *parent) :
    QObject(parent)
{
// nothing to be done on the constructor
}

 /**
this is a virtual method we need to implement this method , most important step
we will create our custom QNetworkAccessManager here and return that
the second important thing we need to do here is to connect the sslErrors signal from
QNetworkAccessManager to our own slot
which will ignore the sslErrors
*/
QNetworkAccessManager* CustomNetworkManagerFactory::create(QObject *parent)
{
    m_networkManager = new QNetworkAccessManager(this);

connect(m_networkManager,SIGNAL(sslErrors(QNetworkReply*,QList<QSslError>)),this,SLOT(onIgnoreSSLErrors(QNetworkReply*,QList<QSslError>)));

    return m_networkManager;
}

/**
Our own slot which is connected to the sslErrors signal of QNetworkAccessManager
When this slot is called using the QNetworkReply object in the parameter we need to call
```


(C) Copyright Nokia 2014. All rights reserved.

```
ignoreSslErrors method of QNetworkReply
*/
void CustomNetworkManagerFactory::onIgnoreSSLErrors(QNetworkReply *reply,
QList<QSslError> error)
{
    reply->ignoreSslErrors(error);
}
```

Once this class is complete we need to make sure that the QDeclarativeEngine uses the custom QDeclarativeNetworkAccessManagerFactory which we have designed above. To do that, please check the implementation of the main.cpp file as shown below:

```
#include "customnetworkmanagerfactory.h"


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

   // declare and intialize your custom network manager factory
    CustomNetworkManagerFactory* myNetworkAccessManagerFactory = new
CustomNetworkManagerFactory(&app);

   // QDeclarativeEngine Interface provided by application template
    QmlApplicationViewer viewer;

    //set our customnetworkmanagerfactory on the declarative engine
    viewer.engine()->setNetworkAccessManagerFactory(myNetworkAccessManagerFactory);

   return app.exec();
}
```

Once the developer successfully completes this then opening https websites will not be a problem.

Summary

This code snippet helps developer to understand the problem with SSL websites and QML webview on Symbian^3 devices and helps them to bypass the SSL errors by implementing our own custom QNetworkAccessManager and passing that to QDeclarativeEngine of QML.

(C) Copyright Nokia 2014. All rights reserved.