УДК 004.056

© 2020 г. **В.Д. Епанешников**, д-р физ.-мат. наук
(Дальневосточный государственный университет путей сообщения, Хабаровск),
И.В. Епанешникова, канд. техн. наук (Хабаровск)

РАСШИРЕНИЕ ЗАЩИТЫ ИНФОРМАЦИОННЫХ СИСТЕМ ЗАЩИТОЙ ИНДИВИДУАЛЬНЫХ WEB-ПРИЛОЖЕНИЙ В ОС LINUX

Возможности класса QHttpServer, используемого для создания серверных web-приложений в операционной системе Linux, были расширены добавлением протокола SSL. Это позволило связывать браузер и серверное WEB-приложение по защищенному кодированному TCP соединению. Проект был реализован в среде Qt Creator 3.3.0 (основан на Qt 5.4.0) операционной системы Fedora-20 (32-битная версия) (коммерческой версией Fedora является Red Hat). Проект также был реализован в среде Qt Creator 4.2.1 (основан на Qt 5.8.0) операционной системы Fedora- 25 (64-битная версия) и в операционной системе AstraLinux.
Ключевые слова: защита информации, серверное web-приложение, протокол SSL.

DOI: 10.22250/isu.2020.63.109-115

Введение

Разработка WEB-приложений находит все большее применение для решения широкого круга технических и научных задач. Это позволяет использовать всю аппаратную и программную мощность сервера, и нет необходимости создавать специальную программу-клиента, так как в этом качестве можно задействовать любой браузер. Создавать такие приложения лучше в среде программирования, имеющей доступ к обширным библиотекам функций, реализующих решение множества прикладных задач. В операционной системе Windows такими средами WEB-программирования являются, например, MS Visual Studio, RAD Studio, позволяющие пользоваться языками C#, C++ и обширными библиотеками функций. В операционной системе Linux такую возможность предоставляет, например, пакет Qt [1] с использованием QT Creator. В стандартную поставку Qt не входит класс, реализующий

функции WEB-сервера на языке C++. Однако существует его реализация с открытым кодом по адресу [2], которая хорошо включается в среду Qt и позволяет создавать web-приложения, удовлетворяющие практическим требованиям. Обмен информацией между браузером и серверным WEB-приложением осуществляется по TCP соединению, которое совершенно не защищено, что по нынешним временам является серьезным недостатком. В среде Internet существует несколько предложений, как защитить соединение QHttpServer и браузера по протоколу SSL, например, [3], однако полной реализации этих идей пока не опубликовано. В настоящей статье представлена практическая реализация использования протокола SSL для защиты информации TCP соединения браузера и QHttpServer.

Постановка задачи и реализация

Использование протокола SSL [4] для защиты TCP соединения предпочтительно потому, что его поддержка уже включена в браузере и остается организовать ее только на стороне QHttpServer. Используя идеи [3, 5], можно произвести замену взаимодействия классов следующим образом. Исходное взаимодействие классов QHttpServer ←→ HttpConnection (здесь используется незащищенное TCP соединение) представлено на рис. 1.

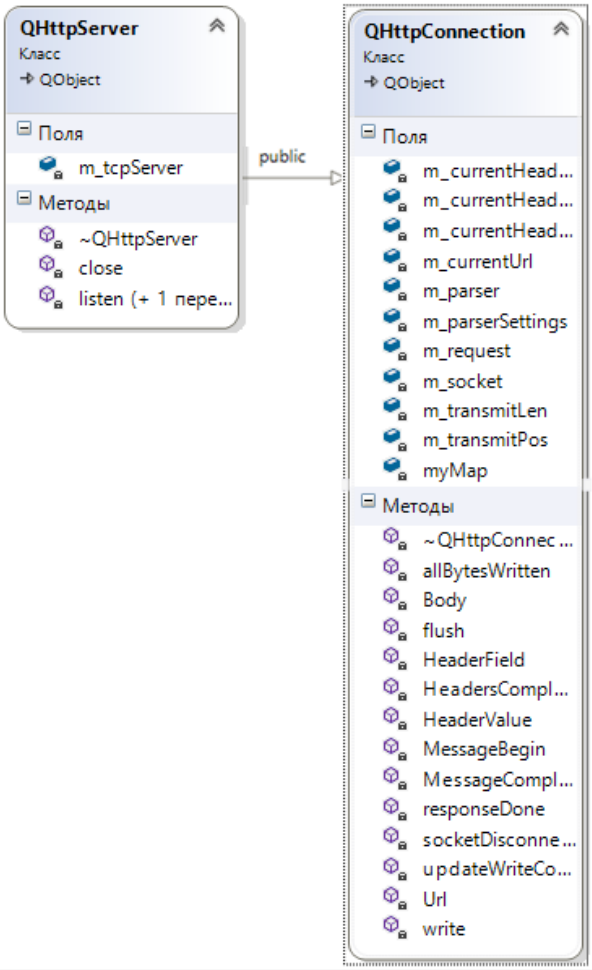

Рис. 1. Упрощенная диаграмма исходных классов.

Эту последовательность классов следует заменить на другую последовательность QHttpServer ←→ QSslServer ←→ QHttpConnection (в классе QSslServer используется QSslSocket), которая представлена на рис. 2.

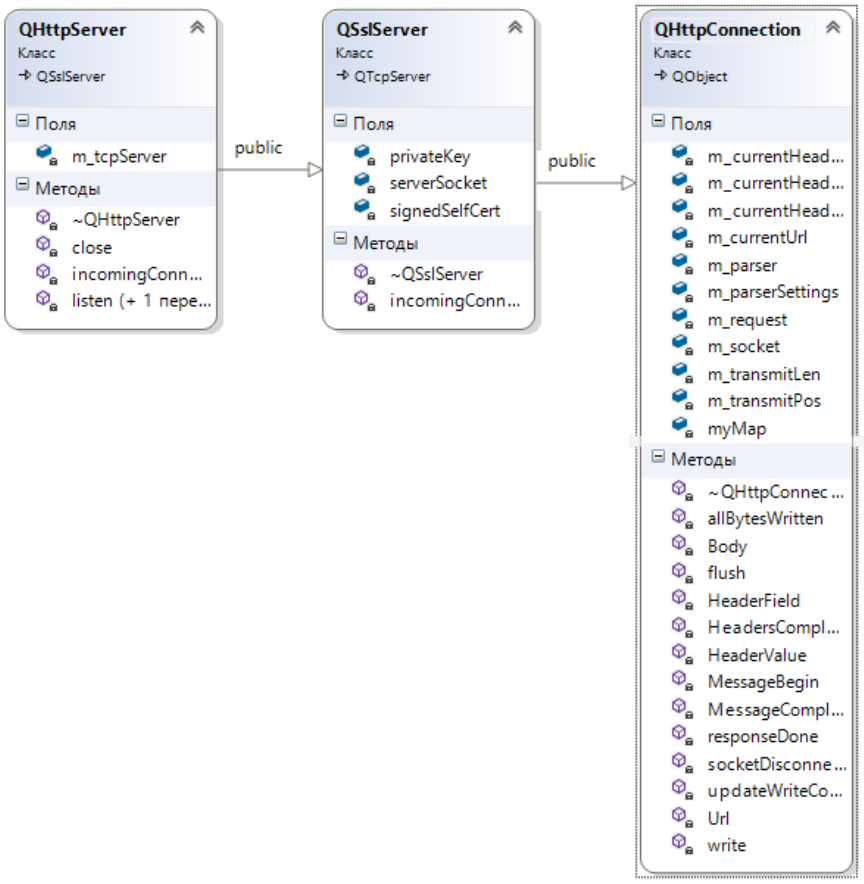

Рис. 2. Упрощенная диаграмма новой последовательности классов.

Для этого необходимо произвести некоторые изменения в исходных файлах [2]. Ниже приведены фрагменты текстов изменяемых файлов (рис. 3 – 5). Все изменения и дополнения выделены жирным шрифтом.

```
#ifndef Q_HTTP_SERVER_FWD
#define Q_HTTP_SERVER_FWD
#include <QHash>
#include <QString>
typedef QHash<QString, QString> HeaderHash;
// QHttpServer
class QHttpServer;
class QHttpConnection;
class QHttpRequest;
class QHttpResponse;
class  QSslServer;
// Qt
class QTcpServer;
class QTcpSocket;
// http_parser
struct http_parser_settings;
struct http_parser;
#endif
```

Рис. 3. Изменения в файле qhttpserverfwd.h

```
#ifndef Q_HTTP_SERVER
#define Q_HTTP_SERVER
#define QHTTPSERVER_VERSION_MAJOR 0
#define QHTTPSERVER_VERSION_MINOR 1
#define QHTTPSERVER_VERSION_PATCH 0
#include "qhttpserverapi.h"
#include "qhttpserverfwd.h"
#include <QObject>
#include <QHostAddress>
#include <QTcpServer>
#include <QSsl>
#include <QSslSocket>
#include <QSslKey>
#include <qsslsocket.h>
class  QSslServer : public QTcpServer
{
   Q_OBJECT
   public:
      QSslServer(QObject *parent=0);
      QSslSocket *serverSocket;


   virtual ~QSslServer();
   public slots:
      void ready();


   private:
      QSslCertificate signedSelfCert;
      QSslKey privateKey;
      void incomingConnection(qintptr socketDescriptor);
};
extern QHash<int, QString> STATUS_CODES;
class QHTTPSERVER_API QHttpServer : public QSslServer
{
   Q_OBJECT
   public:
      QHttpServer(QObject *parent=0);
      QSslServer  *m_tcpServer;
      virtual ~QHttpServer();
      bool listen(const QHostAddress &address = QHostAddress::Any, quint16 port = 80);
      bool listen(quint16 port);
      void close();
   signals:
      void newRequest(QHttpRequest *request, QHttpResponse *response);
   public slots:
      private slots:
      void newConnection();
   private:
      void incomingConnection(qintptr socketDescriptor);
};
#endif
```

Рис. 4. Изменения в файле qhttpserverfwd.h.

```
#include "qhttpserver.h"
#include <QTcpServer>
#include <QTcpSocket>
#include <QVariant>
#include <QDebug>
#include <QSslSocket>
#include "qhttpconnection.h"
#include "bodydata.h"
QHash<int, QString> STATUS_CODES;
QSslServer::QSslServer(QObject *parent)
{
}
QSslServer::~QSslServer()
{
}
……….
bool QHttpServer::listen(const QHostAddress &address, quint16 port)
{
   Q_ASSERT(!m_tcpServer);
   m_tcpServer = new QSslServer(this);
   bool couldBindToPort = m_tcpServer->listen(address, port);
   if (couldBindToPort)
   {
      connect(m_tcpServer, SIGNAL(newConnection()), this, SLOT(newConnection()));
   }
   else
   {
      delete m_tcpServer;
      m_tcpServer = NULL;
   }
   return couldBindToPort;
}
void QSslServer::incomingConnection(qintptr socketDescriptor)
{
   serverSocket = new QSslSocket;
   QFile certFile("/home/evd/crl/localhost.crt");
   certFile.open(QIODevice::ReadOnly);
   signedSelfCert = QSslCertificate (&certFile);
   certFile.close();
   QFile privateFile("/home/evd/crl/localhost.key");
   privateFile.open(QIODevice::ReadOnly);
   privateKey = QSslKey(&privateFile, QSsl::Rsa);
   privateFile.close();
   serverSocket->setPrivateKey(privateKey);
   serverSocket->setLocalCertificate(signedSelfCert);
   if (serverSocket->setSocketDescriptor(socketDescriptor))
   {
      addPendingConnection(serverSocket);
      connect(serverSocket, &QSslSocket::encrypted, this, &QSslServer::ready);
      serverSocket->startServerEncryption();
   }
   else
   {
      delete serverSocket;
   }
}
void QSslServer::ready()
{
}
```

Рис. 5. Изменения в файле qhttpserver.cpp.

Все изменения были осуществлены при создании серверного WEB-приложения, реализующего краткосрочное прогнозирование момента времени и магнитуды будущего землетрясения на основе волновых форм предшествующих землетрясений [6, 7]. Проект был реализован в среде Qt Creator 3.3.0 (основан на Qt 5.4.0) операционной системы Fedora – 20 (32-битная версия) (коммерческой версией Fedora является Red Hat). Были использованы внутренние сертификационные файлы Fedora-20 localhost.crt и localhost.key, которые были скопированы в другой рабочий каталог проекта.

Проект также был реализован в среде Qt Creator 4.2.1 (основан на Qt 5.8.0) операционной системы Fedora-25 (64-битная версия) и в операционной системе AstraLinux (64-битная версия).

На рис. 6 и 7 представлены входные формы проекта в различных браузерах.

Рис. 6. Входная форма проекта в браузере Yandex.

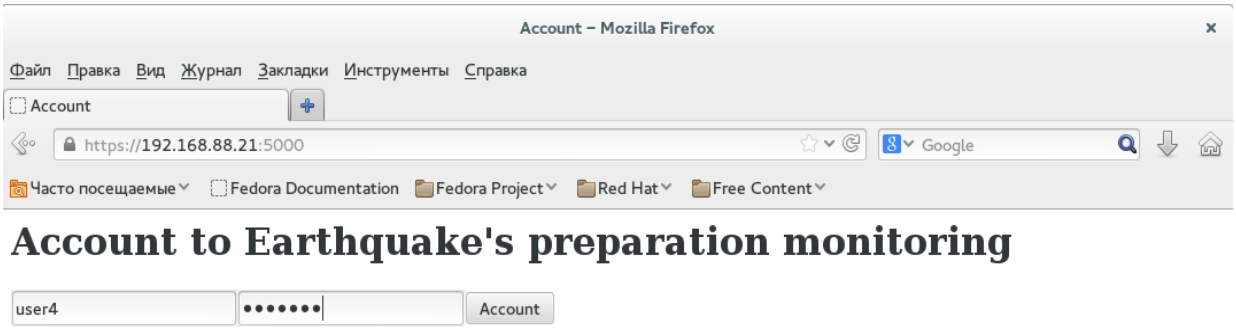

Рис. 7. Входная форма проекта в браузере Firefox.

При вызове проекта необходимо соглашаться на использование ненадежного сертификата. Факт передачи по сети кодированных сообщений проверялся снифером WireShark [8].

Заключение

Web-приложение, созданное на основе класса QHttpServer, является мини-WEB-сервером, у которого оставлены только методы, необходимые для обслуживания самого web-приложения. Это снижает риск успешных атак на такой мини-WEB-сервер по сравнению с полновесными WEB-серверами.

Расширение возможностей класса QHttpServer включением протокола SSL повышает надежность и защищенность разработанных на его основе серверных

WEB-приложений в операционной системе Linux, исключая возможность перехвата логинов и паролей, а также шифрованием всего трафика при взаимодействии браузера и серверного WEB-приложения.

ЛИТЕРАТУРА

1. URL: https://www.qt.io/ru/ (время обращения – ноябрь 2019 г.)
2. URL: https://github.com/nikhilm/qhttpserver/ (время обращения – ноябрь 2019 г.)
3. URL: https://github.com/nikhilm/qhttpserver/issues/33 (время обращения – ноябрь 2019 г.)
4. *Фороузан Б.А*. Криптография и безопасность сетей. – М.: Интернет-Университет информационных технологий: Бином. Лаборатория знаний. 2010.
5. URL: https://doc-snapshots.qt.io/qt5-5.9/qsslsocket.html (время обращения – ноябрь 2019 г.)
6. *Пат.* 2558277 РФ. Способ краткосрочного прогнозирования локальной магнитуды землетрясения / В.Д. Епанешников, И.В. Епанешникова // Официальный бюл. «Изобретения. Полезные модели». – 2015. – № 21. – С. 1-9.
7. *Епанешникова И.В.* Защищенное ПО SSL серверное WEB-приложение мониторинга краткосрочных предвестников землетрясений В ОС LINUX. № 2019613705, 21.03.2019 Бюл. № 4. URL:http://www1.fips.ru/wps/PA_FipsPub/res/BULLETIN/PrEVM/2019/04/20/ INDEX.HTM.
8. URL: https://www.wireshark.org (время обращения – ноябрь 2019 г.)

Статья представлена к публикации членом редколлегии Е.Л. Ереминым.

E-mail:
Епанешников Владимир Дмитриевич – evd_44@mail.ru;
Епанешникова Ирина Владимировна – evd_44@mail.ru.