# OpenSSL 的 C/S 安全通信

陈 波 张碧云

**摘 要** 本文介绍了 OpenSSL 在 Win32 平台上的安装配置及 SSL 编程接口，并给出了一个基于 OpenSSL 的客户端、服务器安全通信的程序。

**关键词** 安全套接字层协议，传输层安全，OpenSSL

## 一、安全套接字层 SSL 协议

现代信息系统及网络通信系统常常遇到中断、截获、篡改和伪造等一些威胁。

面对网络中的诸多安全威胁，为通信提供强大的安全成为必需之举。通信安全的中心内容就是保证信息在通道内的保密性、完整性和可认证性(真实性)。

SSL(Secure Socket Layer，安全套接字层)正是在两台机器之间提供安全通信的协议。SSL 是 Netscape 公司于 1994 年提出的，目前 SSL3.0 得到了业界广泛认可，已成为事实上的标准。TLS(Transport Layer Security，传输层安全)协议是 IETF 的 TLS 工作组在 SSL v3.0 基础之上提出的，目前版本是 1.0。TLS v1.0 可看作 SSL v3.1，和 SSL v3.0 的差别不大。

SSL 协议应用在传输层和应用层之间，采用对称加密或非对称加密技术、消息摘要技术、数字证书等密码技术实现通信安全的三个方面，即通过认证连接两端，传输经过加密的数据，并且能确保数据是完整没有被篡改的。

1. 保密性

SSL 支持多种加密套件，这些加密套件指定一组不同强度供连接使用的算法。SSL 协议通过握手过程完成服务器和客户端所使用加密套件的协商，随后通信中的数据传输均采用协商好的加密算法。

2. 完整性

利用 Hash 函数计算出所要发送信息的摘要值，保证对方可以验证所接受的数据是否被篡改。消息摘要算法也在握手过程中商定。

3. 可认证性

SSL 协议默认支持对服务器的认证，使用户可以确定连接另一端的身份，实现方法是通过检查数字证书链确定该服务器是否在自己的信任列表里，从而确定该服务器是否可信任。另外，SSL 协议也支持对客户端的认证。

## 二、开源工具包 OpenSSL

目前，除了 SSL 标准的提出者 Netscape 公司的实现外，OpenSSL 是一个非常优秀的实现 SSL/TLS 的著名开源软件包，它实现了 SSL v2.0、SSL v3.0 和 TLS v1.0。OpenSSL 采用 C 语言作为开发语言，具有良好的跨平台性能。OpenSSL 支持 Linux、Windows、BSD、MAC 等平台，具有广泛的实用性。

OpenSSL 在结构上可分为三层：最底层是各种密码算法的实现；中间层是密码算法的抽象接口，它对各种算法按对称、非对称、Hash 算法进行分类后提供一组接口；最上层是围绕密码算法的 PKCS(公钥加密标准，Public Key Cryptographic Standard)的实现。OpenSSL 整个软件包可以分成 3 个主要的功能部分：crypto 密码算法库、SSL 协议库、OpenSSL 命令行工具程序。

OpenSSL 主要提供以下功能：各类密钥以及密钥参数的生成和格式转换；使用各种加密算法进行数据加密；证书请求、证书生成和签发以及证书其他相关标准的转换；消息摘要算法及其相关编码的实现；SSL 服务器和 SSL 客户端安全通信的实现。

与 OpenSSL 相比，Microsoft 也提供了一个功能完善的密码算法库 CryptoAPI，包含了各种密码算法、密钥管理及证书管理功能，但是其不开放源代码和仅基于 Windows 平台的缺点，使得技术人员不能彻底地了解它，不能根据自己的需要进行修改和跨平台使用，此外其作为美国公司的安全产品受到密码算法的出口限制也是让我们不能接受的。

David 完成的Crypto++ 也是一个不错的开源加密算法库，它使用 C++ 编写，但是它仅仅是 OpenSSL 的一个子集，没有实现如 X.509 标准和 SSL 协议。

从上面的比较我们可以认识到 OpenSSL 是开发安全通信软件的首选。

## 三、Win32 系统下的 OpenSSL 安装配置

1. 安装 OpenSSL

从 OpenSSL 的官方网站 http://www.openssl.org 下载最新的源代码。将下载的压缩文件包解压(本文解压在 D:\，下同)。其中主要的文件夹中包含的内容介绍见下表。

下载、安装 Win32 版的 Perl(ActivePerl)、NASM。OpenSSL 需要使用 perl 进行一些配置。OpenSSL 同样支持 MASM，如果使用它，安装命令要稍作更改，具体操作可阅读 OpenSSL 根目录下的安装说明文件 install.w32。

设置环境变量，包括 perl 命令文件夹、openssl 开发包的解压文件夹、NASM 工具文件夹和 VC 命令文件夹。建议用系统属性来设置环境变量，因为 set path 是临时的，当 Console 窗口关闭后，path 就失效了。

>set Path=D:\Perl\bin\;D:\openssl-0.9.7;D:\nasm-0.98.39;C:\Program Files\Microsoft Visual Studio\VC98\Bin

配置 Win32 下的编译环境，Configure 文件在 openssl 根目录下。

>perl Configure VC-WIN32

继续编译，nmake 为 VC 命令工具

>ms\do_nasm

>nmake -f ms\ntdll.mak

等待几分钟后编译完成，命令行界面自动消失。编译结果保存在 d:\openssl\out32dll\目录下面。

查看编译结果，测试配置是否成功

>cd out32dll

>..\ms\test

2. 配置 VC 开发环境

OpenSSL 编译后会生成 inc32 目录，其中包括了所有 OpenSSL 声明文件，编译后还会在 out32dll 目录中生成 libeay32.lib、libeay32.dll、ssleay32.lib、ssleay32.dll 库文件。OpenSSL 支持多种操作系统与开发工具，Win32 环境下支持 Visual C++、Borland C 和 GNU C (Mingw32 or Cygwin)，其他开发环境下的配置可参照文档 install.w32。将 OpenSSL 的 API 声明拷贝到 VC 的 include 目录。将 OpenSSL 的 API 库文件 libeay32.lib、ssleay32.lib 拷贝到 VC 的 lib 目录。将 OpenSSL 的 API 动态链接库文件 libeay32.dll、ssleay32.dll 拷贝到 Win32 系统的 system32 目录。

3. 生成服务器证书

在 out32dll 目录下，生成用于加密服务器证书的密钥：

openssl genrsa -des3 -out priv.key 1024

生成服务器自验证证书(-config 用于指定配置文件的路径)：

openssl req -new -x509 -key priv.key -out cert.pem -config ..\apps\openssl.cnf

表 OpenSSL 部分文件夹的功能说明

| 文件夹名 | 功能描述 |
|---|---|
| Crypto | 存放 OpenSSL 所有加密算法源码文件和相关标注，是 OpenSSL 中最重要的文件夹 |
| SSL | 存放 OpenSSL 中 SSL 协议各个版本和 TLS v1.0 协议源码文件，包含了 OpenSSL 协议库的所有内容 |
| Apps | 存放 OpenSSL 中所有应用程序的源码文件，如 CA、X.509 |
| Doc | 存放 OpenSSL 中所有的使用说明文档，包括：应用程序说明文档、加密算法库 API 说明文档以及 SSL 协议 API 说明文档 |
| Demos | 存放一些基于 OpenSSL 的应用程序示例，演示怎样使用 OpenSSL 的一些功能 |
| Include | 存放使用 OpenSSL 库时需要的头文件 |
| Test | 存放 OpenSSL 自身功能测试程序的源码文件 |

随后将加密服务器证书的密钥文件 priv.key 和服务器自验证证书文件 cert.pem 拷入服务器端程序所在的文件夹中，再将 cert.pem 拷入客户端程序所在的文件夹中，就可以在程序中用于对服务器进行认证了。

## 四、安全通信程序分析

1. 程序主要流程

本通信示例程序能够完成客户机与服务器(C/S)间的信息加密传输、完整性验证以及客户机对服务器身份的认证等功能，程序流程如图 1 所示。

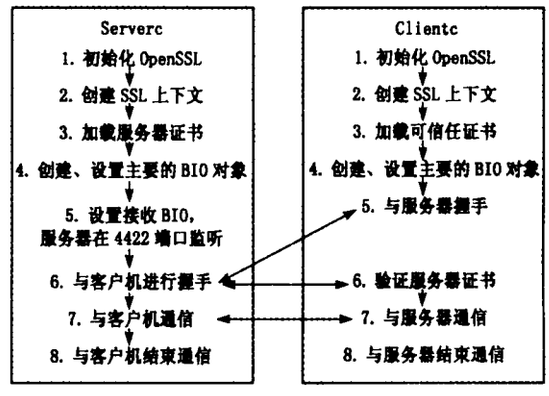


图 1 程序主要流程

2. 服务器端源程序关键步骤分析

本程序的结构与使用 WinSock 的 C/S 网络编程结构类似。

为安全连接进行设置需要有一个类型为 SSL_CTX 的指针来保存 SSL 信息。可以通过调用 SSL 方法函数 SSL_CTX_new 来创建这个结构。还需要另一个 SSL 类型的指针 ssl 来保持 SSL 连接结构(这是短时间就能完成的一些连接所必需的)。以后还可以用该 ssl 指针来检查连接信息或设置其他 SSL 参数。

在创建上下文结构之后，须加载一个服务器证书。这其中

包括3个主要工作：安装回调函数，提供证书私钥密码；加载服务器证书，该证书包含服务器公钥，是发送给客户机的；加载服务器证书私钥。

在建立安全连接之前创建一个指向BIO对象的指针。这类似于在标准C中为文件流创建FILE指针。BIO用来处理包括文件和套接字在内的各种类型的通信。需要使用3个BIO对象：bio指向主要的BIO对象；abio是接受连接使用的BIO，用来接收到达的连接；out指向服务器与客户机通信的对象。

此处对BIO对象的设置与客户机连接使用的BIO设置稍有不同。客户机连接使用BIO_new_ssl_connect建立，此处设置是使用BIO_new_ssl加上两个参数建立的：一个指向SSL_CTX对象的指针和一个标记，这个标记告诉OpenSSL要创建哪种BIO对象：0用于服务器，1用于客户机。

然后设置接收BIO，服务器在4422端口监听。BIO_new_accept为服务器连接创建BIO，它只需要一个参数，就是监听的端口。BIO_set_accept_bios将一个安全BIO链接到这个接收BIO上。即将前面创建的主要的BIO对象bio连接到该接收BIO对象abio上。在WinSock编程中，用accept函数等待客户端的连接，OpenSSL中则是BIO_do_accept函数实现。在等待之前，必须要调用BIO_do_accept两次，第一次调用是设置BIO来接收到达连接，第二次调用则是真正等待。

服务器使用BIO_pop来响应连接。之后，使用BIO_do_handshake函数与客户端进行握手。BIO_set_conn_hostname用于设置服务器的主机名和端口，BIO_do_connect为客户机连接创建BIO。

服务器与客户端握手成功后，就可在一个加密信道上进行通信了。这样可阻止中间人攻击。

最后，调用BIO_free_all或BIO_reset关闭连接，具体调用哪一个方法取决于是否重用BIO，同时必须在结束应用程序之前调用SSL_CTX_free释放SSL上下文结构。服务器端的核心代码如下：

下面是完整的服务器端源代码：

```
int password_callback(char * buf, int size, int rwflag, void *
userdata)
{  //加载证书的函数没有提供证书私钥密码的功能,
   //OpenSSL为获得该私钥密码提供了一种回调机制
   printf("Callback function called\n");
  strcpy(buf, "njnusky"); //私钥密码为njnusky
  return strlen(buf);
}
int main(int argc, char * * argv)
{
  SSL_CTX * ctx;  //SSL上下文指针
  SSL * ssl;  //SSL指针
   BIO * bio,  * abio,  * out;
  int  ( * callback)  (char * , int, int, void * ) = & pass-
word_callback;
int n;
char r[1024];
  puts("服务器运行在安全模式下.\n\n");
  //1. 初始化OpenSSL - - - - - - - - - - - - - - -
  SSL_library_init();
  SSL_load_error_strings();
  ERR_load_BIO_strings();
  ERR_load_SSL_strings();
  OpenSSL_add_all_algorithms();
  //2. 创建SSL上下文 - - - - - - - - - - - - - - -
  printf("创建SSL上下文...");
 ctx = SSL_CTX_new(SSLv23_server_method());
 //3. 加载服务器证书 - - - - - - - - - - - - - - -
printf("\n加载证书...\n");
SSL_CTX_set_default_passwd_cb(ctx, callback);
   if( !  SSL_CTX_use_certificate_file(ctx,  "  cert.pem",
SSL_FILETYPE_PEM))
    printf("加载服务器证书失败...\n");
 !  SSL_CTX_use_PrivateKey_file(ctx,  "  priv.key",
SSL_FILETYPE_PEM))
 printf("加载服务器私钥失败...\n");
  //4. 创建、设置主要的BIO对象 - - - - - - - - - -
  printf("创建、设置主要的BIO对象...\n");
  bio = BIO_new_ssl(ctx, 0);
  BIO_get_ssl(bio, & ssl);
  SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);
  //5. 设置接收BIO,服务器在4422端口监听 - - - -
  printf("\n服务器在端口4422监听...\n");
  abio = BIO_new_accept("4422");
  BIO_set_accept_bios(abio, bio);
printf("等待客户端连接...\n\n");
BIO_do_accept(abio);
BIO_do_accept(abio);
  //6. 与客户机进行握手 - - - - - - - - -
  out = BIO_pop(abio);
  printf("与客户端进行握手...\n\n");
if(BIO_do_handshake(out) <0)
    printf("握手失败...\n\n");
    //7. 与客户机通信 - - - - - - - - - - - - - - -
printf("接受客户端数据:\n");
for(;;)
  {
    n = BIO_read(out, r, 1023);
    if(n < = 0) break;
    r[n] = 0;
    printf("Client: %s", r);
    if(strcmp(r, "Hello, I am a client.\n") = = 0)
        BIO_puts(out, "Hello, I am waiting...\n");
    else if(strcmp(r, "quit\n") = = 0)
        break;
```

```
        else
            BIO_puts(out, "Ok, I accept! \n");
    }
        BIO_puts(out, "Bye! \n");
    BIO_flush(out);
    //8. 与客户机结束通信 - - - - - - - - - - - - -
    BIO_free_all(out); BIO_free_all(bio); BIO_free_all(abio);
     SSL_CTX_free(ctx);
     return 0;
}
```

**3. 客户端源程序分析**

**客户端源代码中，第 1～5 步的操作类似于服务器端的步骤。第 6 步是客户端验证服务器证书的步骤，客户机根据可信任证书(库)来核实服务器向客户机发送一个证书，以确保它没有过期，且是可信任的。一个可信任证书(库)在第 3 步中加载。握手成功并且验证服务器证书有效后，即与服务器建立安全信道，就可以进行加密通信。客户端核心源代码如下：**

```
int main(int argc, char * * argv)
{
    BIO * bio; //指向与服务器连接的 BIO 对象
    SSL * ssl;
    SSL_CTX * ctx; //指向 SSL 上下文
    int p;
     const char * request = "Hello, I am a client. \n";
    char r[1024];
     char content[1024];
    //1. 初始化 OpenSSL - - - - - - - - - -
    ERR_load_BIO_strings();
    SSL_load_error_strings();
    OpenSSL_add_all_algorithms();
    //2. 创建 SSL 上下文,为安全连接进行设置 - - - -
    ctx = SSL_CTX_new(SSLv23_client_method());
    //3. 加载可信任证书 - - - - - - - -
     SSL_CTX_load_verify_locations(ctx, "cert.pem", NULL);
    //4. 创建、设置主要的 BIO 对象 - - - - - - - - -
     bio = BIO_new_ssl_connect(ctx);
    BIO_get_ssl(bio, & ssl);
    SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);
    //5. 与服务器握手 - - - - - - - - - - - - - - -
     BIO_set_conn_hostname(bio, "localhost: 4422");
    if(BIO_do_connect(bio) < = 0)
    {
      //此处可添加连接或握手失败处理....
    }
     //6. 验证服务器证书 - - - - - - - - - - - - -
     if(SSL_get_verify_result(ssl) ! = X509_V_OK)
    {
      printf(" 证书无效! \n"); //  此处可添加证书无效处理
....
    }
    else
        printf("证书有效! \n");
     //7. 与服务器通信 - - - - - - - - - - - - - -
    BIO_write(bio, request, strlen(request));
    for(; ; )
    {
      p = BIO_read(bio, r, 1023);
      if(p < = 0) break;
      r[p] = 0;
      printf("Server: %s", r);
       if(strcmp(r, "Bye! \n") = = 0)
            break;
        scanf("%s", content);
        strcat(content, "\n");
        BIO_puts(bio, content);
    }
     //8. 与服务器结束通信 - - - - - - - - - - - - -
    BIO_free_all(bio);
    SSL_CTX_free(ctx);
    return 0;
}
```

## 五、结语

**本文给出了 OpenSSL 的 C/S 安全通信的程序，可以完成 C/S 间的信息加密传输、完整性验证以及客户机对服务器身份的认证等功能。**

**本程序在 VC + + 6.0 环境下编译成功。经过加密的通信可以有效的阻止中间人攻击，图 2 为使用 Wireshark(原 Ethereal)包嗅探工具捕获的服务器、客户机之间的加密通信数据。**

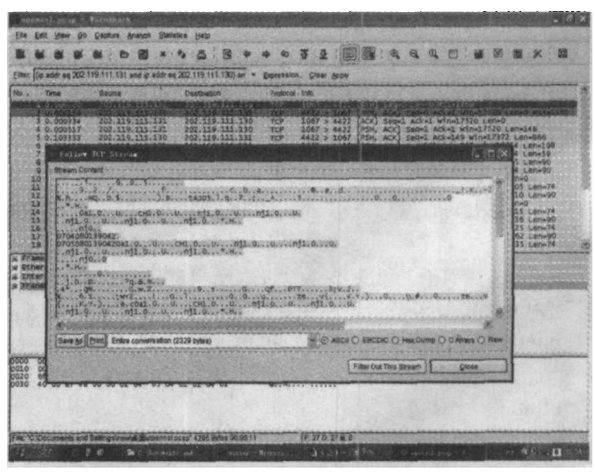

**图 2 嗅探工具捕获的服务器、客户机之间的加密通信数据**

(收稿日期：2007 年 5 月 26 日)