OpenSSL API 的文档有些含糊不清。因为还没有多少关于 OpenSSL 使用的教程，所以对初学者来说，在 应用程序中使用它可能会有一些困难。那么怎样才能使用 OpenSSL 实现一个基本的安全连接呢？ 本教程将帮助您解决这个问题。

学习如何实现 OpenSSL 的困难部分在于其文档的不完全。不完全的 API 文档通常会妨碍开发人员 使用该 API，而这通常意味着它注定要失败。但 OpenSSL 仍然很活跃，而且正逐渐变得强大。这是为什么？

OpenSSL 是用于安全通信的最著名的开放库。在 google 中搜索“SSL library”得到的返回结果中， 列表最上方就是 OpenSSL。它诞生于 1998 年，源自 Eric Young 和 Tim Hudson 开发的 SSLeay 库。其他 SSL 工具包包括遵循 GNU General Public License 发行的 GNU TLS，以及 Mozilla Network Security Services（NSS）（请参阅本文后面的 [参考资料](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#Resources) ，以获得 其他信息）。

那么，是什么使得 OpenSSL 比 GNU TLS、Mozilla NSS 或其他所有的库都优越呢？许可是一方面因素 （请参阅 [参考资料](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#Resources)）。此外，GNS TLS（迄今为止）只支持 TLS v1.0 和 SSL v3.0 协议，仅此而已。

Mozilla NSS 的发行既遵循 Mozilla Public License 又遵循 GNU GPL，它允许开发人员进行选择。 不过，Mozilla NSS 比 OpenSSL 大，并且需要其他外部库来对库进行编译，而 OpenSSL 是完全 自包含的。与 OpenSSL 相同，大部分 NSS API 也没有文档资料。Mozilla NSS 获得了 PKCS #11 支持，该支持可以用于诸如智能卡这样的加密标志。OpenSSL 就不具备这一支持。

**先决条件**

要充分理解并利用本文，您应该：

* 精通 C 编程。
* 熟悉 Internet 通信和支持 Internet 的应用程序的编写。

并不绝对要求您熟悉 SSL ，因为稍后将给出对 SLL 的简短说明；不过，如果您希望得到详细论述 SSL 的文章的链接，请参阅 [参考资料部分](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#Resources)。拥有密码学方面的知识固然好，但这 并不是必需的。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**什么是 SSL？**

SSL 是一个缩写，代表的是 Secure Sockets Layer。它是支持在 Internet 上进行安全通信的 标准，并且将数据密码术集成到了协议之中。数据在离开您的计算机之前就已经被加密，然后只有 到达它预定的目标后才被解密。证书和密码学算法支持了这一切的运转，使用 OpenSSL，您将 有机会切身体会它们。

理论上，如果加密的数据在到达目标之前被截取或窃听，那些数据是不可能被破解的。不过， 由于计算机的变化一年比一年快，而且密码翻译方法有了新的发展，因此，SSL 中使用的加密协议 被破解的可能性也在增大。

可以将 SSL 和安全连接用于 Internet 上任何类型的协议，不管是 HTTP、POP3，还是 FTP。还可以用 SSL 来保护 Telnet 会话。虽然可以用 SSL 保护任何连接，但是不必对每一类连接都使用 SSL。 如果连接传输敏感信息，则应使用 SSL。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**什么是 OpenSSL？**

OpenSSL 不仅仅是 SSL。它可以实现消息摘要、文件的加密和解密、数字证书、数字签名 和随机数字。关于 OpenSSL 库的内容非常多，远不是一篇文章可以容纳的。

OpenSSL 不只是 API，它还是一个命令行工具。命令行工具可以完成与 API 同样的工作， 而且更进一步，可以测试 SSL 服务器和客户机。它还让开发人员对 OpenSSL 的能力有一个 认识。要获得关于如何使用 OpenSSL 命令行工具的资料，请参阅 [参考资料](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#Resources)部分。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**您需要什么**

首先需要的是最新版本的 OpenSSL。查阅参考资料部分，以确定从哪里可以获得最新的可以自己编译的源代码， 或者最新版本的二进制文件（如果您不希望花费时间来编译的话）。不过，为了安全起见， 我建议您下载最新的源代码并自己编译它。二进制版本通常是由第三方而不是由 OpenSSL 的开发人员来编译和发行的。

一些 Linux 的发行版本附带了 OpenSSL 的二进制版本，对于学习如何使用 OpenSSL 库来说，这足够了；不过， 如果您打算去做一些实际的事情，那么一定要得到最新的版本，并保持该版本一直是最新的。

对于以 RPM 形式安装的 Linux 发行版本（Red Hat、Mandrake 等），建议您通过从发行版本制造商那里获得 RPM 程序包来更新您的 OpenSSL 发行版本。出于安全方面的原因，建议您使用 最新版本的发行版本。如果您的发行版本不能使用最新版本的 OpenSSL，那么建议您只覆盖库文件，不要覆盖 可执行文件。OpenSSL 附带的 FAQ 文档中包含了有关这方面的细节。

还要注意的是，OpenSSL 并没有在所有的平台上都获得官方支持。虽然制造商已经尽力使其能够跨平台兼容， 但仍然存在 OpenSSL 不能用于您的计算机 和/或 操作系统的可能。请参阅 OpenSSL 的 Web 站点（ [参考资料](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#Resources) 中 的链接），以获得关于哪些平台可以得到支持的信息。

如果想使用 OpenSSL 来生成证书请求和数字证书，那么必须创建一个配置文件。在 OpenSSL 程序包 的 apps 文件夹中，有一个名为 openssl.cnf 的 可用模板文件。我不会对该文件进行讨论，因为这不在本文要求范围之内。不过，该模板文件有一些非常好的注释，而且如果 在 Internet 上搜索，您可以找到很多讨论修改该文件的教程。

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**头文件和初始化**

本教程所使用的头文件只有三个：ssl.h、bio.h 和 err.h。它们都位于 openssl 子目录中，而且都是开发您的项目 所必需的。要初始化 OpenSSL 库，只需要三个代码行即可。清单 1 中列出了所有内容。其他的头文件 和/或 初始化函数可能 是其他一些功能所必需的。

**清单 1. 必需的头文件**

|  |
| --- |
| /\* OpenSSL headers \*/  #include "openssl/bio.h"  #include "openssl/ssl.h"  #include "openssl/err.h"  /\* Initializing OpenSSL \*/  SSL\_load\_error\_strings();  ERR\_load\_BIO\_strings();  OpenSSL\_add\_all\_algorithms(); |

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**建立非安全连接**

不管连接是 安全的还是不安全的，OpenSSL 都使用了一个名为 BIO 的抽象库来处理包括文件和套接字在内的各种类型的通信。您还可以将 OpenSSL 设置成为一个过滤器，比如用于 UU 或 Base64 编码的过滤器。

在这里对 BIO 库进行全面说明有点麻烦，所以我将根据需要一点一点地介绍它。首先， 我将向您展示如何建立一个标准的套接字连接。相对于使用 BSD 套接字库，该操作需要 的代码行更少一些。

在建立连接（无论安全与否）之前，要创建一个指向 BIO 对象的指针。这类似于在标准 C 中 为文件流创建 FILE 指针。

**清单 2. 指针**

|  |
| --- |
| BIO \* bio; |

**打开连接**

创建新的连接需要调用 BIO\_new\_connect 。您可以在同一个调用中同时 指定主机名和端口号。也可以将其拆分为两个单独的调用：一个是创建连接并设置主机名的 BIO\_new\_connect 调用，另一个是设置端口号的 BIO\_set\_conn\_port （或者 BIO\_set\_conn\_int\_port ）调用。

不管怎样，一旦 BIO 的主机名和端口号都已指定，该指针会尝试打开连接。没有什么可以影响它。如果创建 BIO 对象时遇到问题，指针将会是 NULL。为了确保连接成功，必须执行 BIO\_do\_connect 调用。

**清单 3. 创建并打开连接**

|  |
| --- |
| bio = BIO\_new\_connect("hostname:port");  if(bio == NULL)  {  /\* Handle the failure \*/  }  if(BIO\_do\_connect(bio) <= 0)  {  /\* Handle failed connection \*/  } |

在这里，第一行代码使用指定的主机名和端口创建了一个新的 BIO 对象，并以所示风格对该对象进行 格式化。例如， 如果您要连接到 www.ibm.com 的 80 端口，那么该字符串将是 www.ibm.com:80 。调用 BIO\_do\_connect 检查连接是否成功。如果出错，则返回 0 或 -1。

**与服务器进行通信**

不管 BIO 对象是套接字还是文件，对其进行的读和写操作都是通过以下两个函数来完成的： BIO\_read 和 BIO\_write 。 很简单，对吧？精彩之处就在于它始终如此。

BIO\_read 将尝试从服务器读取一定数目的字节。它返回读取的字节数、 0 或者 -1。在受阻塞的连接中，该函数返回 0，表示连接已经关闭，而 -1 则表示连接出现错误。在非阻塞连接的情况下，返回 0 表示没有可以获得的数据，返回 -1 表示连接出错。可以调用BIO\_should\_retry 来确定是否可能重复出现该错误。

**清单 4. 从连接读取**

|  |
| --- |
| int x = BIO\_read(bio, buf, len);  if(x == 0)  {  /\* Handle closed connection \*/  }  else if(x < 0)  {  if(! BIO\_should\_retry(bio))  {  /\* Handle failed read here \*/  }  /\* Do something to handle the retry \*/  } |

BIO\_write 会试着将字节写入套接字。它将返回实际写入的 字节数、0 或者 -1。同 BIO\_read ，0 或 -1 不一定表示错误。BIO\_should\_retry 是找出问题的途径。如果需要重试写操作，它必须 使用和前一次完全相同的参数。

**清单 5. 写入到连接**

|  |
| --- |
| if(BIO\_write(bio, buf, len) <= 0)  {  if(! BIO\_should\_retry(bio))  {  /\* Handle failed write here \*/  }  /\* Do something to handle the retry \*/  } |

**关闭连接**

关闭连接也很简单。您可以使用以下两种方式之一来关闭连接： BIO\_reset 或 BIO\_free\_all 。如果您还需要重新使用对象，那么请使用第一种方式。 如果您不再重新使用它，则可以使用第二种方式。

BIO\_reset 关闭连接并重新设置 BIO 对象的内部状态，以便可以重新使用连接。如果要在整个应用程序中使用同一对象，比如使用一台安全的聊天 客户机，那么这样做是有益的。该函数没有返回值。

BIO\_free\_all 所做正如其所言：它释放内部结构体，并释放 所有相关联的内存，其中包括关闭相关联的套接字。如果将 BIO 嵌入于一个类中，那么应该在类的 析构函数中使用这个调用。

**清单 6. 关闭连接**

|  |
| --- |
| /\* To reuse the connection, use this line \*/  BIO\_reset(bio);  /\* To free it from memory, use this line \*/  BIO\_free\_all(bio); |

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**建立安全连接**

现在需要给出建立安全连接需要做哪些事情。惟一要改变的地方就是建立并进行连接。其他所有内容都是相同的。

安全连接要求在连接建立后进行握手。在握手过程中，服务器向客户机发送一个证书， 然后，客户机根据一组可信任证书来核实该证书。它还将检查证书，以确保它没有过期。要 检验证书是可信任的，需要在连接建立之前提前加载一个可信任证书库。

只有在服务器发出请求时，客户机才会向服务器发送一个证书。该过程叫做客户机认证。使用证书， 在客户机和服务器之间传递密码参数，以建立安全连接。尽管握手是在建立连接之后才进行的，但是客户机或服务器可以在任何时刻请求进行一次新的握手。

[参考资料](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#Resources) 部分中列出的 Netscasp 文章 和 RFC 2246 ，对握手以及建立安全连接的其他方面的知识进行了更详尽的论述。

**为安全连接进行设置**

为安全连接进行设置要多几行代码。同时需要有另一个类型为 SSL\_CTX 的指针。该结构保存了一些 SSL 信息。您也可以利用它通过 BIO 库建立 SSL 连接。可以通过使用 SSL 方法函数调用 SSL\_CTX\_new 来创建这个结构，该方法函数通常是SSLv23\_client\_method 。

还需要另一个 SSL 类型的指针来保持 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。

**清单 7. 设置 SSL 指针**

|  |
| --- |
| SSL\_CTX \* ctx = SSL\_CTX\_new(SSLv23\_client\_method());  SSL \* ssl; |

**加载可信任证书库**

在创建上下文结构之后，必须加载一个可信任证书库。这是成功验证每个证书所必需的。如果 不能确认证书是可信任的，那么 OpenSSL 会将证书标记为无效（但连接仍可以继续）。

OpenSSL 附带了一组可信任证书。它们位于源文件树的 certs 目录中。 不过，每个证书都是一个独立的文件 —— 也就是说，需要单独加载每一个证书。在 certs 目录下，还有一个存放过期证书的子目录。试图加载这些证书将会出错。

如果您愿意，可以分别加载每一个文件，但为了简便起见，最新的 OpenSSL 发行版本的可信任证书 通常存放在源代码档案文件中，这些档案文件位于名为“TrustStore.pem”的单个文件中。如果已经有了一个可信任证书库， 并打算将它用于特定的项目中，那么只需使用您的文件替换清单 8 中的“TrustStore.pem”（或者使用 单独的函数调用将它们全部加载）即可。

可以调用 SSL\_CTX\_load\_verify\_locations 来加载可信任证书库文件。这里要用到 三个参数：上下文指针、可信任库文件的路径 和文件名，以及证书所在目录的路径。必须指定可信任库文件或证书的目录。 如果指定成功，则返回 1，如果遇到问题，则返回 0。

**清单 8. 加载信任库**

|  |
| --- |
| if(! SSL\_CTX\_load\_verify\_locations(ctx, "/path/to/TrustStore.pem", NULL))  {  /\* Handle failed load here \*/  } |

如果打算使用目录存储可信任库，那么必须要以特定的方式命名文件。OpenSSL 文档清楚 地说明了应该如何去做，不过，OpenSSL 附带了一个名为 c\_rehash 的工具， 它可以将文件夹配置为可用于 SSL\_CTX\_load\_verify\_locations 的 路径参数。

**清单 9. 配置证书文件夹并使用它**

|  |
| --- |
| /\* Use this at the command line \*/  c\_rehash /path/to/certfolder  /\* then call this from within the application \*/  if(! SSL\_CTX\_load\_verify\_locations(ctx, NULL, "/path/to/certfolder"))  {  /\* Handle error here \*/  } |

为了指定所有需要的验证证书，您可以根据需要命名任意数量的单独文件或文件夹。您还可以同时指定 文件和文件夹。

**创建连接**

将指向 SSL 上下文的指针作为惟一参数，使用 BIO\_new\_ssl\_connect 创建 BIO 对象。还需要获得指向 SSL 结构的指针。在本文中，只将该指针用于 SSL\_set\_mode 函数。而这个函数是用来设置 SSL\_MODE\_AUTO\_RETRY 标记的。使用这个选项进行设置，如果服务器突然希望进行 一次新的握手，那么 OpenSSL 可以在后台处理它。如果没有这个选项，当服务器希望进行一次新的握手时， 进行读或写操作都将返回一个错误，同时还会在该过程中设置 retry 标记。

**清单 10. 设置 BIO 对象**

|  |
| --- |
| bio = BIO\_new\_ssl\_connect(ctx);  BIO\_get\_ssl(bio, & ssl);  SSL\_set\_mode(ssl, SSL\_MODE\_AUTO\_RETRY); |

设置 SSL 上下文结构之后，就可以创建连接了。主机名是使用 BIO\_set\_conn\_hostname 函数 设置的。主机名和端口的指定格式与前面的相同。该函数还可以打开到主机的连接。为了确认已经成功打开连接，必须 执行对 BIO\_do\_connect 的调用。该调用还将执行握手来建立安全连接。

**清单 11. 打开安全连接**

|  |
| --- |
| /\* Attempt to connect \*/  BIO\_set\_conn\_hostname(bio, "hostname:port");  /\* Verify the connection opened and perform the handshake \*/  if(BIO\_do\_connect(bio) <= 0)  {  /\* Handle failed connection \*/  } |

连接建立后，必须检查证书，以确定它是否有效。实际上，OpenSSL 为我们完成了这项任务。如果证书有致命的 问题（例如，哈希值无效），那么将无法建立连接。但是，如果证书的问题并不是致命的（当它已经过期 或者尚不合法时），那么仍可以继续使用连接。

可以将 SSL 结构作为惟一参数，调用 SSL\_get\_verify\_result 来查 明证书是否通过了 OpenSSL 的检验。如果证书通过了包括信任检查在内的 OpenSSL 的内部检查，则返回 X509\_V\_OK。如果有地方出了问题，则返回一个错误代码，该代码被记录在命令行工具的 verify 选项下。

应该注意的是，验证失败并不意味着连接不能使用。是否应该使用连接取决于验证结果和安全方面的考虑。例如， 失败的信任验证可能只是意味着没有可信任的证书。连接仍然可用，只是需要从思想上提高安全意识。

**清单 12. 检查证书是否有效**

|  |
| --- |
| if(SSL\_get\_verify\_result(ssl) != X509\_V\_OK)  {  /\* Handle the failed verification \*/  } |

这就是所需要的全部操作。通常，与服务器进行通信都要使用 BIO\_read 和 BIO\_write 。并且只需调用 BIO\_free\_all 或 BIO\_reset ，就可以关闭 连接，具体调用哪一个方法取决于是否重用 BIO。

必须在结束应用程序之前的某个时刻释放 SSL 上下文结构。可以调用 SSL\_CTX\_free 来释放该结构。

**清单 13. 清除 SSL 上下文**

|  |
| --- |
| SSL\_CTX\_free(ctx); |

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**错误检测**

显然 OpenSSL 抛出了某种类型的错误。这意味着什么？首先，您需要得到错误代码本身； ERR\_get\_error 可以完成这项任务；然后，需要将错误代码转换为错误 字符串，它是一个指向由 SSL\_load\_error\_strings 或 ERR\_load\_BIO\_strings 加载到内存中的永久字符串的指针。 可以在一个嵌套调用中完成这项操作。

表 1 略述了从错误栈检索错误的方法。清单 24 展示了如何打印文本字符串中的最后一个 错误信息。

**表 1. 从栈中检索错误**

|  |  |
| --- | --- |
| ERR\_reason\_error\_string | 返回一个静态字符串的指针，然后可以将字符串显示在屏幕上、写入文件，或者以任何您希望的方式进行处理 |
| ERR\_lib\_error\_string | 指出错误发生在哪个库中 |
| ERR\_func\_error\_string | 返回导致错误的 OpenSSL 函数 |

**清单 14. 打印出最后一个错误**

|  |
| --- |
| printf("Error: %s\n", ERR\_reason\_error\_string(ERR\_get\_error())); |

您还可以让库给出预先格式化了的错误字符串。可以调用 ERR\_error\_string 来 得到该字符串。该函数将错误代码和一个预分配的缓冲区作为参数。而这个缓冲区必须是 256 字节长。如果参数 为 NULL，则 OpenSSL 会将字符串写入到一个长度为 256 字节的静态缓冲区中，并返回指向该缓冲区的 指针。否则，它将返回您给出的指针。如果您选择的是静态缓冲区选项，那么在下一次调用ERR\_error\_string 时，该缓冲区会被覆盖。

**清单 15. 获得预先格式化的错误字符串**

|  |
| --- |
| printf("%s\n", ERR\_error\_string(ERR\_get\_error(), NULL)); |

您还可以将整个错误队列转储到文件或 BIO 中。可以通过 ERR\_print\_errors 或 ERR\_print\_errors\_fp 来实现这项操作。队列是以可读格式被转储的。第一个函数将队列发送到 BIO ，第二个函数将队列发送到 FILE 。 字符串格式如下（引自 OpenSSL 文档）：

[pid]:error:[error code]:[library name]:[function name]:[reason string]:[file name]:[line]:[optional text message]

其中， [pid] 是进程 ID， [error code] 是一个 8 位十六进制代码， [file name] 是 OpenSSL 库中的源代码文件， [line] 是源文件中的行号。

**清单 16. 转储错误队列**

|  |
| --- |
| ERR\_print\_errors\_fp(FILE \*);  ERR\_print\_errors(BIO \*); |

[**回页首**](http://www.ibm.com/developerworks/cn/linux/l-openssl.html#ibm-pcon)

**开始做吧**

使用 OpenSSL 创建基本的连接并不困难，但是，当试着确定该如何去做时，文档可能是一个小障碍。本文向您介绍了一些基本概念，但 OpenSSL 还有很多灵活之处有待发掘，而且 您还可能需要一些高级设置，以便项目能够充分利用 SSL 的功能。

本文中有两个样例。一个样例展示了到 http://www.verisign.com/ 的非安全连接，另一个则展示了到 http://www.verisign.com/ 的安全 SSL 连接。两者都是连接到服务器并下载其主页。它们没有进行 任何安全检查，而且库中的所有设置都是默认值 —— 作为本文的一部分，应该只将这些用于教学目的。

在任何支持的平台上，源代码的编译都应该是非常容易的，不过我建议您使用最新版本的 OpenSSL。在撰写本文时，OpenSSL 的最新版本是 0.9.7d。