

OpenSSL 談 SSL Programming 的抽象化

Jim Huang (黃敬群)

最後更新：Apr 6, 2005

筆者維護 gaim[1] 的 OpenSSL 支援[2]，雖然因為授權議題無法整合到 gaim 正式發行版本，不過維護這個分支倒是對於 SSL 應用有了些體驗。gaim 的 SSL plugin 主要用於 MSN Messenger 在處理 protocol 初期的認證部份，gaim 作為一個多重 IM (即時訊息傳遞) 的應用程式，其 plugin 設計相當可取，所以這裡從 OpenSSL 與 gaim 整合作出發，探討運作原理與進一步的抽象化。

SSL 的基本操作

儘管 SSL 有相當多模式，但是簡單來說，還是歸屬於 Stream I/O 的操作，而在 gaim ssl plugin interface 的設計則要求 plugin 的實做必須提供以下操作：

- ssl_init
- ssl_uninit
- ssl_connect_cb
- ssl_close
- ssl_read
- ssl_write

相當明顯的，前兩項就是作所謂初始化與終結的動作，而我們的重點將置放於後四者。ssl_connect_cb 的設計用意就是將某個已經開啓的 socket connection 「聯繫」到底層 SSL 實做，當然本文是用 OpenSSL 作介紹。必須注意的是，SSL connection 過程必須維護 SSL context 的資訊，相當 SSL 有效的生命期，為了與 socket programming 能接軌，所以才會有如此的設計。一旦 SSL context 建立成功，ssl_close / ssl_read / ssl_write 這三者就只是底層 SSL 實做的 wrapper，包裝特定 API。

探討 OpenSSL 應用

於是乎，ssl_connect_cb 的實做就成為相當重要的部份，筆者引述所維護實做的設計，當然方法不只一種，但是因為 OpenSSL 提供足夠的 API，所以實做也相當簡單。首先看看 function prototype: (plugins/ssl/ssl-openssl.c)

```
/*
 * ssl_openssl_connect_cb
 *
 * given a socket, put an openssl connection around it.
 */
static void ssl_openssl_connect_cb(
    gpointer data,
```

```
gint source,  
GaimInputCondition cond);
```

這裡的 data 是原始 gaim 傳遞資料 (raw data)，筆者將定義一個 struct 來保存 OpenSSL 特有的資訊：

```
typedef struct {  
    SSL *ssl;  
    SSL_CTX *ssl_ctx;  
} GaimSslOpensslData;
```

注意，SSL 指的是 "SSL Connection Pool"，而 SSL_CTX 是 SSL 連線建立一併維護的 SSL context，對 OpenSSL 來說已經包裝足夠的資訊，所以只要兩個項目即可。如果用比較陽春的 SSL 實做，比方說 MatrixSSL[3]，就必須自己維護 Connection Pool 所需的資訊，程式碼幾乎是兩倍以上長度。

爾後我們將適度予以轉型並宣告一個內部保存資訊專用的項目：

```
GaimSslConnection *gsc = (GaimSslConnection *)data;  
GaimSslOpensslData *openssl_data;
```

上述的 gint source 是 File Descriptor (以下簡稱 fd)，筆者將作驗證並把 SSL 連線與此 fd 相關連：

```
if (source < 0)  
    return;  
gsc->fd = source;
```

接下來就是重點了：

```
openssl_data->ssl_ctx = SSL_CTX_new(SSLv23_client_method());  
if (openssl_data->ssl_ctx == NULL) {  
    gaim_debug_error("openssl", "SSL_CTX_new failed\n");  
    if (gsc->error_cb != NULL) {  
        gsc->error_cb(  
            gsc,  
            GAIM_SSL_HANDSHAKE_FAILED,  
            gsc->connect_cb_data);  
    }  
    gaim_ssl_close(gsc);  
    return;
```

```
}
```

這個程式碼片段中，我們建立了新的 SSL_CTX 物件，倘若建立失敗，就會做出 Handshaking 的錯誤，而如果順利建立 SSL context 後，我們接著就要建立合法的 SSL connection，而程式碼片對幾乎跟上面一樣，只不過第一行改成以下：

```
openssl_data->ssl = SSL_new(openssl_data->ssl_ctx);
```

SSL connection 接受剛剛產生的 SSL context 物件作為參數，然後此新物件就將可用以工作了，我們看看：

```
if (SSL_set_fd(openssl_data->ssl, source) == 0) {
    gaim_debug_error("openssl", "SSL_set_fd failed\n");
    if (gsc->error_cb != NULL) {
        gsc->error_cb(
            gsc,
            GAIM_SSL_HANDSHAKE_FAILED,
            gsc->connect_cb_data);
    }
    gaim_ssl_close(gsc);
    return;
}
```

注意到呼叫 OpenSSL API 中的 SSL_set_fd 呼叫，這將合法的 fd 與剛剛產生的 SSL connection 物件彼此關聯，如果沒有錯誤的話，我們幾乎就成功了。就這樣嗎？需要試通以確認：

```
if (SSL_connect(openssl_data->ssl) <= 0) {
    gaim_debug_error("openssl", "SSL_connect failed\n");
    if (gsc->error_cb != NULL) {
        gsc->error_cb(
            gsc,
            GAIM_SSL_HANDSHAKE_FAILED,
            gsc->connect_cb_data);
    }
    gaim_ssl_close(gsc);
    return;
}
```

如果首次試通的傳回長度大於 0 的話，就代表整個 connect_cb 已經大功告成。

而 read 與 write 操作就直接對應到 OpenSSL API 的 SSL_read 與 SSL_write 這兩個 API，需要注意的是 close 操作必須先後釋放 SSL connection 與 SSL context，分別對應到 SSL_shutdown / SSL_free 與 SSL_CTX_free 等 API。

SSL Programming 抽象化設計

OpenSSL 在眾多 open source SSL library 來說，是相當完整，也提供許多便利性，但是為了許多考量，我們有可能需要轉換到其他 SSL 實做，比方說為了更小的 memory footprint，有可能需要改用 MatrixSSL，但是正如上文所提，MatrixSSL 並無內建 UNIX fd 對 SSL connection 的對應，所以必須在 connect_cb 作些功夫。但是事實上 gaim SSL plugin interface 所定義的項目幾乎就可以完成抽象的 SSL Programming，這也是為何 gaim 可以同時支援許多 protocol，也提供許多實做方式的原因。

[1] <http://gaim.sourceforge.net/>

[2] 這些特別的 gaim 分支下載位址：<http://jserv.sayya.org/gaim/>

[3] <http://www.matrixssl.org/>