Communication sécurisée avec *SSL*

Mise en œuvre d'OpenSSL
Application à HTTPS

Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

Propos

- **Limiter le risque lié à la communication**
  - Confidentialité : données lisibles par quiconque ?
    - Utiliser des algorithmes de chiffrement
  - Intégrité : données modifiées pendant le transport ?
    - Utiliser des algorithmes de hachage (condensé)
  - Authentification : dialogue avec l'entité attendue ?
    - Obtenir un document officiel identifiant l'interlocuteur
- **Préserver les protocoles applicatifs**
  - Ne pas nécessiter une réécriture complète des applications
  - Une couche supplémentaire dans la pile de protocoles (usage très similaire à celui de la couche de transport)
- **Principes de fonctionnement discutés dans le module *ASR***
  - Seule la mise en œuvre est vue ici

Origine et implémentation

- **Spécifications et développement initial**
  - *Netscape* en 1994, introduction dans les serveurs et navigateurs
  - Version 3.1 désormais désignée par *TLS*
  - À l'origine pour sécuriser les sites *web* (*https*)
    - Administration, commerce, paiement en ligne ...
- **Implémentation courante**
  - *OpenSSL* : implémentation libre, la plus largement répandue
    - Une bibliothèque
    - La commande en ligne `openssl`
      (*"couteau suisse"* de la cryptographie)
  - Il en existe d'autres
  - Utilisation plus large que les services *web* (*smtps*, *ssh* ...)

Principe de mise en œuvre

- **L'application établit une connexion *TCP***
  - Serveur : `socket()`, `bind()`, `listen()`, `accept()`
  - Client : `socket()`, `connect()`
- **Cette connexion est confiée à *SSL* et lui sert de support**
  - À chaque extrémité, le descripteur est encapulé dans un objet `SSL`
  - Une négociation de la connexion sécurisée (*handshake*) à lieu
    - Algorithmes de chiffrement asymétrique et symétrique
    - Algorithmes de hachage et éventuellement de compression
    - Échange de *certificats*, de clefs ...
- **L'*API SSL* offre à l'application des fonctions d'entrée/sortie**
  - Utilisation similaire à `read()`/`write()`
  - Chiffrement, hachage, compression selon ce qui a été négocié
  - Transparent pour l'application, surcoût en volume/traitement limité

Mise en œuvre en ligne de commande

- **L'utilitaire** `openssl`
  - `$ openssl` *`command options`*
  - De très nombreuses commandes liées à la cryptographie
  - Client/serveur génériques pour interagir *"à la main"*
- **Quelques exemples pour une mise en œuvre minimale**
  - Client *SSL* générique ($\simeq$ client `nc` avec chiffrement)

    ```
    $ openssl s_client -host hostname -port port
    ```

  - Serveur *SSL* générique ($\simeq$ serveur `nc` avec chiffrement)

    ```
    $ openssl s_server -accept port -cert cert.pem -key key.pem
    ```

  - Générer un *certificat* auto-signé (`-x509`) sans pass-phrase (`-nodes`)

    ```
    $ openssl req -x509 -nodes -newkey rsa:1024 \
                  -keyout key.pem -out cert.pem
    ```

Mise en œuvre en ligne de commande

▷ Le programme client générique

```
$ openssl s_client -host iroise.enib.fr -port 443
CONNECTED(00000003)
...
-----BEGIN CERTIFICATE-----
MIIDkDCCAvmgAwIBAgIJALjom3jqP3AIMA0GCSqGSIb3DQEBBQUAMIGNMQswCQYD
...
-----END CERTIFICATE-----
subject=/C=FR/ST=Bretagne/L=BREST/O=ENI de BREST/OU=CRI/CN=iroise.enib.fr/emailAddress=iroise@enib.fr
issuer=/C=FR/ST=Bretagne/L=BREST/O=ENI de BREST/OU=CRI/CN=iroise.enib.fr/emailAddress=iroise@enib.fr
---
...
---
GET / HTTP/1.0
Host: iroise.enib.fr:443

HTTP/1.1 403 Forbidden
Date: Mon, 13 Aug 2007 13:03:14 GMT
Server: Apache/2.2.2 (Fedora)
Accept-Ranges: bytes
Content-Length: 3931
Connection: close
Content-Type: text/html; charset=UTF-8

<html> ... </html>
```

Mise en œuvre en ligne de commande

▷ Le programme serveur générique

```
$ openssl s_server -accept 9443 -cert cert.pem -key key.pem
Using default temp DH parameters
Using default temp ECDH parameters
ACCEPT
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMBBAIAOQQgzRKKpUnQWNiKHMet44cQlPa3t5i+GfvpQIF+aB1RUAgE
...
-----END SSL SESSION PARAMETERS-----
Shared ciphers: ...
CIPHER is DHE-RSA-AES256-SHA
GET /dummy.html HTTP/1.0
User-Agent: Wget/1.10.2
Accept: */*
Host: localhost:9443

HTTP/1.0 200 OK

<html><body>Hello</body></html>
DONE
shutdown accept socket
shutting down SSL
CONNECTION CLOSED
```

Mise en œuvre en ligne de commande

▷ Génération du *certificat* auto-signé du serveur

- ◇ Le *Common Name* (*CN*) doit correspondre au nom du serveur
- ◇ nb : Pour une mise en œuvre complète, la gestion des paires de clefs et des *certificats* mérite discussion (cf module *ASR*)

```
$ openssl req -x509 -nodes -newkey rsa:1024 -keyout key.pem -out cert.pem
Generating a 1024 bit RSA private key
...
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Brest
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ENIB
Organizational Unit Name (eg, section) []:CERV
Common Name (eg, YOUR name) []:dummy.enib.fr
Email Address []:dummy@enib.fr
```

Mise en œuvre de l'*API*

▷ **Réaliser un programme utilisant** *OpenSSL*

◇ Inclusion de `<openssl/ssl.h>` et de `<openssl/err.h>`

◇ Compilation, édition de liens avec `pkg-config`

```
$ cc -c proc.c `pkg-config openssl --cflags`
$ cc -o prog proc.o `pkg-config openssl --libs`
```

▷ **Initialisation du programme**

```
SSL_CTX * ctx;
SSL_library_init();
SSL_load_error_strings();
OpenSSL_add_all_algorithms();
ctx=SSL_CTX_new(SSLv23_method());
```

▷ **Terminaison du programme**

```
SSL_CTX_free(ctx);
```

Mise en œuvre de l'*API*

▷ Squelette d'un programme client minimal

```
SSL_CTX * ctx;
int fd,r;
SSL * ssl;
SSL_library_init(); SSL_load_error_strings(); OpenSSL_add_all_algorithms();
ctx=SSL_CTX_new(SSLv23_method());

fd=connectTcp(getIpAddress("iroise.enib.fr"),443); /* client TCP (netUtils.h) */
if(fd==-1) { perror("connectTcp"); return -1; }
ssl=SSL_new(ctx);                                   /* creer une connexion SSL */
SSL_set_mode(ssl,SSL_MODE_AUTO_RETRY);           /* renegotiation automatique */
SSL_set_fd(ssl,fd);            /* la connexion SSL utilisera la connexion TCP */
r=SSL_connect(ssl);                  /* demarrer le handshake avec le serveur */
if(r!=1) { fprintf(stderr,"SSL_connect: %s\n",
                          ERR_error_string(ERR_get_error(),NULL)); return -1; }
/* ... utiliser la connexion SSL pour dialoguer avec le serveur */
SL_free(ssl);                     /* detruire la connexion en fin de dialogue */
close(fd);

SSL_CTX_free(ctx);            /* detruire le contexte SSL en fin de programme */
```

Mise en œuvre de l'API

▷ Squelette d'un programme serveur (1/2)

```
SSL_CTX * ctx;
unsigned long addr;
int port;
int sock,fd,r;
SSL * ssl;
SSL_library_init();
SSL_load_error_strings();
OpenSSL_add_all_algorithms();
SSL_CTX_new(SSLv23_method());

/* fournir le certificat et la clef-privee du serveur */
SSL_CTX_use_certificate_file(ctx,"cert.pem",SSL_FILETYPE_PEM);
SSL_CTX_use_PrivateKey_file(ctx,"key.pem",SSL_FILETYPE_PEM);
if(!SSL_CTX_check_private_key(ctx))  /* certificat/clef-privee compatibles ? */
  { fprintf(stderr,"SSL_CTX_check_private_key: %s\n",
                  ERR_error_string(ERR_get_error(),NULL)); return -1; }
```

Mise en œuvre de l'*API*

▷ Squelette d'un programme serveur (2/2)

```
sock=listenTcp(9443);                                  /* serveur TCP (netUtils.h) */
if(sock==-1) { perror("listenTcp"); return -1; }

fd=acceptTcp(sock,&addr,&port);    /* attendre une connexion TCP (netUtils.h) */
if(fd==-1) { perror("acceptTcp"); return -1; }
ssl=SSL_new(ctx);                                       /* creer une connexion SSL */
SSL_set_mode(ssl,SSL_MODE_AUTO_RETRY);             /* renegotiation automatique */
SSL_set_fd(ssl,fd);              /* la connexion SSL utilisera la connexion TCP */
r=SSL_accept(ssl);                       /* demarrer le handshake avec le client */
if(r!=1) { fprintf(stderr,"SSL_accept: %s\n",
                           ERR_error_string(ERR_get_error(),NULL)); return -1; }
/* ... utiliser la connexion SSL pour dialoguer avec le client */
SL_free(ssl);                         /* detruire la connexion en fin de dialogue */
close(fd);

close(sock);
SSL_CTX_free(ctx);              /* detruire le contexte SSL en fin de programme */
```

Mise en œuvre de l'*API*

- **Fonction d'écriture similaire à l'appel système** `write()`
  - `int SSL_write(SSL * ssl,const void * buf,int count);`
  - Les erreurs peuvent être dues à *TCP* ou *SSL*

```
SSL * ssl= ...
char * data= ...
unsigned int dataSize= ...
int r=SSL_write(ssl,data,dataSize);
if(r>0) { /* r octets de data envoyes */ }
else
  {
  if(SSL_get_error(ssl,r)==SSL_ERROR_SYSCALL) { /* TCP, consulter errno */ }
  else { fprintf(stderr,"SSL_write: %s\n",ERR_error_string(ERR_get_error(),NULL)); }
  }
```

Mise en œuvre de l'*API*

- **Fonction de lecture similaire à l'appel système** `read()`
  - `int SSL_read(SSL * ssl,void * buf,int count);`
  - Les erreurs peuvent être dues à *TCP* ou *SSL*
  - La fin de fichier peut être due à *TCP* ou *SSL*

```
SSL * ssl= ...
char buffer[BUFFER_SIZE];
int r=SSL_read(ssl,buffer,BUFFER_SIZE);
if(r>0) { /* r octets recus dans buffer */ }
else
  {
  if(!r||(SSL_get_error(ssl,r)==SSL_ERROR_ZERO_RETURN)) { /* EOF TCP ou SSL */ }
  else if(SSL_get_error(ssl,r)==SSL_ERROR_SYSCALL) { /* TCP, consulter errno */ }
  else { fprintf(stderr,"SSL_read: %s\n",ERR_error_string(ERR_get_error(),NULL)); }
  }
```

Mise en œuvre de l'*API*

- **Comportement vis-à-vis de la scrutation passive**
  - Volume de données de `SSL_read()` $\not\equiv$ volume de données de `read()`
    - Utilisation de tampons en interne pour le déchiffrement
  - Si `select()` sur le descripteur de fichier puis `SSL_read()`
    - Risque de blocage alors que des données sont prêtes dans le tampon
  - Consulter la quantité prête dans le tampon avec `SSL_pending()`

```
int fd= ... ; /* connexion TCP */
SSL * ssl= ... ; /* connexion SSL reposant sur fd */

int maxFd=-1;
fd_set rdSet;
struct timeval tv0={0,0}; /* preparation d'un eventuel timeout de duree nulle */
struct timeval * ptv=(struct timeval *)0;           /* pas de timeout initial */
if(SSL_pending(ssl)) ptv=&tv0;       /* si donnees deja pretes, pas d'attente */
else FD_SET_MAX(fd,&rdSet,maxFd);        /* sinon scrutation de fd necessaire */
RESTART_SYSCALL(r,select(maxFd+1,&rdSet,(fd_set *)0,(fd_set *)0,ptv));
if(r==-1) { perror("select"); return -1; }
if(SSL_pending(ssl)||FD_ISSET(fd,&rdSet)) { /* pret pour SSL_read() sur ssl */ }
```

Mise en œuvre de l'*API*

▷ **Utilisation dans un programme** *multi-threads*

◇ *OpenSSL* n'est pas dépendant d'un bibliothèque de *threads* spécifique

◇ L'application doit fournir les moyens de synchronisation adaptés

```
pthread_mutex_t * ssl_locks; /* l'ensemble des verrous utiles a SSL */
void ssl_locking_cb(int mode,int n,const char * file,int line)
{ (void)file; (void)line; /* variables utiles au debuggage */
if(mode&CRYPTO_LOCK) { pthread_mutex_lock(ssl_locks+n); }
else { pthread_mutex_unlock(ssl_locks+n); }
}
unsigned long ssl_id_cb(void) { return (unsigned long)pthread_self(); }

/* lors de l'initialisation du contexte SSL ... */
ssl_locks=(pthread_mutex_t *)malloc(CRYPTO_num_locks()*sizeof(pthread_mutex_t));
for(i=CRYPTO_num_locks();i--;)          /* creer tous les verrous necessaires */
  { pthread_mutex_init(ssl_locks+i,(pthread_mutexattr_t *)0); }
CRYPTO_set_id_callback(&ssl_id_cb); /* fournir nos fonctions d'identification */
CRYPTO_set_locking_callback(&ssl_locking_cb);       /* et de (de)verrouillage */
```

Bilan intermédiaire

- **La connexion en elle-même est relativement sécurisée**
  - Tout le chiffrement dépend du *certificat* et de la clef-privée du serveur
  - Seuls le client et le serveur peuvent déchiffrer la communication
  - Des segments *TCP* capturés renferment des données incompréhensibles
  - Leur rejeu est inefficace (utilisation de numéros de séquence)
- **Est-on certain de s'adresser au bon serveur ?**
  - Pour l'instant on fait entièrement confiance au serveur !!!
  - Le *certificat* du serveur doit être vérifié par le client
    - Signé par une autorité de certification connue ou auto-signé mais connu
    - Conforme au nom du serveur
- **Discuté dans le module *ASR***

Mise en œuvre de l'*API*

▷ Vérification du *certificat* du serveur par le client (1/2)

- ◇ *Certificat* signé par une autorité de certification connue ?
- ◇ *Common-name* du *certificat* identique au nom du serveur ?

```
SSL_CTX * ctx;
int r;
SSL * ssl;
X509 * cert;
char * hostname= ... /* nom du serveur auquel le client veut se connecter */
/* ... initialisation du contexte SSL */

/* cacert.pem contient les certificats des autorites de certification connues */
/* (voir http://curl.haxx.se/docs/caextract.html ) */
if(!SSL_CTX_load_verify_locations(ctx,"cacert.pem",NULL))
  { fprintf(stderr,"SSL_CTX_load_verify_locations: %s\n",
                  ERR_error_string(ERR_get_error(),NULL)); }

/* ... connexion au serveur */
r=SSL_connect(ssl);
if(r==-1) { "SSL_connect: %s\n", ERR_error_string(ERR_get_error(),NULL)); return -1; }
```

Mise en œuvre de l'API

▷ Vérification du *certificat* du serveur par le client (2/2)

```
r=SSL_get_verify_result(ssl);
if(r!=X509_V_OK) { fprintf(stderr,"!!! Warning !!! Certificate not trusted\n"); }
cert=SSL_get_peer_certificate(ssl);
if(!cert)
  { fprintf(stderr,"SSL_get_peer_certificate: %s\n",
                  ERR_error_string(ERR_get_error(),NULL)); }
else
  {
  char commonName[0x100]="";
  X509_NAME_get_text_by_NID(X509_get_subject_name(cert),NID_commonName,
                            commonName,0x100);
  if(strcmp(commonName,hostName))
    { fprintf(stderr,"!!! Warning !!! Common name `%s' != host name `%s'\n",
                     commonName,hostName); }
  X509_free(cert);
  }
```

Mise en œuvre de l'*API*

▷ **Vérification d'un *certificat* auto-signé par le client**

◇ Le *certificat* est rejetté par `SSL_get_verify_result()`

○ Si ce *certificat* est vu pour la premiere fois
→ Avertir et mémoriser ce *certificat* pour ce serveur

○ S'il est différent de ce qui a été mémorisé pour ce serveur
→ Avertir et mettre fin à la connexion (*MITM*) !!!

◇ Une démarche semblable est utilisée par ***ssh***

◇ On fait confiance à la première connexion au serveur !!!

◇ Mémoriser/comparer les *certificats* avec des condensés

○ L'exemple ci-dessous produit un condensé ***MD5*** du *certificat*

○ `certDigestSize` octets dans `certDigest`

```
unsigned char certDigest[EVP_MAX_MD_SIZE];
unsigned int certDigestSize;
X509 * cert=SSL_get_peer_certificate(ssl);
...
X509_digest(cert,EVP_md5(),certDigest,&certDigestSize);
```

Cas pratique : *HTTPS*

▷ **Connexion directe du client au serveur**

◇ Les client/serveur reprennent les squelettes précédents

◇ Le serveur écoute sur un port dédié (443)

◇ Le dialogue chiffré est le même que celui qui a lieu en clair pour *HTTP*

▷ **Connexion à travers un *proxy***

◇ Le client se connecte au **proxy** et lui envoie **en clair**

`CONNECT` *`host`*`:`*`port`* `HTTP/1.`*`X`* *(1.0 ou 1.1)*
`Host:` *`host`*`:`*`port`*

(ligne vide → fin de l'entête)

◇ Le *proxy* se connecte au serveur indiqué et répond **en clair** au client

`HTTP/1.`*`X`* `200 Connection established` *(1.0 ou 1.1)*
(ligne vide → fin de l'entête)

Cas pratique : *HTTPS*

- **Connexion à travers un *proxy***
  - Le client met en place une connexion *SSL* dans sa connexion *TCP*
  - Le serveur fait de même (connexion sur son port 443)
  - Le *proxy* est relié au client et au serveur par deux connexions *TCP*
  - Il relaye *"aveuglément"* les données **chiffrées** entre elles
    - Dialogue chiffré identique au dialogue direct en clair pour *HTTP*
    - Il ne peut effectuer aucun filtrage sur le contenu
    - Il ne peut pas mettre en *cache* les données
  - Il peut toujours effectuer certains traitements
    - Interdire la visite de certains sites (*blacklist*)
    - Constituer des journaux d'activité (*logs*)
    - Fermer les connexions qui lui semblent trop inactives !!!

Bilan

- **La connexion en elle-même est relativement sécurisée**
  - L'écoute du traffic est inutile
- **On peut être certain de s'adresser au bon serveur**
  - En n'acceptant que les *certificats* en bonne et due forme
  - En ayant une liste d'autorités de certification à jour
  - En ayant une liste de *certificats* auto-signés bien tenue
- **!!! Repose sur la vigilance de l'application et de l'utilisateur !!!**
  - Prise de risque à la première apparition d'un *certificat* auto-signé
  - Une seule fois peut suffire pour divulger son mot de passe ***ssh*** !
  - L'utilisateur doit prendre en compte les avertissements de l'application
- **Sujet et *API* très vastes**
  - Le serveur peut également demander un *certificat* au client
  - Énormément de choses non traitées ici ...