Laboratorul 13 - Protocoale de securitate. utilizarea programatica

Utilizare programatica

Protocolul SSL (Secure Socket Layer - https://www.ssl.com/faqs/faq-what-is-ssl/) a aparut ca urmare a necesitatii de a asigura conexiuni sigure in Internet, in stiva de protocoale fiind situat intre nivelul TCP si nivelul aplicatie. Protocolul asigura autentificarea mutuala a celor doua entitati care comunica, confidentialitatea comunicarii si protectia integritatii datelor. Una dintre cele mai des intalnite aplicatii ale SSL-ului este HTTPS (HTTP securizat), care consta din protocolul HTTP utilizat peste SSL.

SSL si conexiunile securizate pot fi folosite pentru orice tip de protocol pe Internet, HTTP, POP3 sau FTP. SSL poate fi folosit si pentru a securiza sesiuni de Telnet. Orice tip de conexiune poate fi securizat in teorie folosind SSL, dar ar trebui sa fie folosit numai pentru conexiuni care vor folosi date ce trebuie protejate.

OpenSSL ofera mai mult decat SSL, fiind o biblioteca vasta de functii ce permite lucrul cu message digests, encriptari si decriptari de fisiere, certificate digitale, semnaturi digitale. Deasemenea OpenSSL este nu numai un API, ci si un utilitar ce poate fi folosit din linia de comanda care prezinta aceleasi functionalitati expuse de API permitind in plus si testarea serverelor si a clientilor SSL.

Pentru a instala si uneltele de dezvoltare (inclusiv bibliotecile openssl).

```
sudo apt-get install libssl-dev
```

Urmatoarele fisere header bio.h, ssl.h, err.h vor fi folosite pe parcurs. Daca libssl-dev nu e instalat, importul fisierelor header de mai jos va da erori la compilare.

```
/* OpenSSL headers */
#include "openssl/bio.h"
#include "openssl/ssl.h"
#include "openssl/err.h"
```

De asemenea sunt necesare urmatoarele linii de cod pentru a initializa libraria OpenSSL:

```
/* Initializare OpenSSL */
SSL_library_init ();
SSL_load_error_strings();
ERR_load_BIO_strings();
OpenSSL_add_all_algorithms();
```

OpenSSL foloseste o biblioteca numita BIO pentru a asigura comunicarea atit securizata cat si nesecurizata. Pentru a crea o conexiune fie ea securizata sau nu, un pointer pentru un obiect de tip BIO trebuie creat similar cu crearea unui pointer la FILE (in C). Crearea unei noi conexiuni se face cu *BIO_new_connect()*, specificand hostname si portul. Daca a aparut vreo eroare la crearea obiectului BIO, functia va intoarce NULL. Totodata se va incerca sa se deschida conexiunea. Pentru a verifica daca acest lucru s-a realizat cu succes se apeleaza *BIO_do_connect()* care returneaza 0 (OK) sau -1 (eroare).

```
BIO * bio;
bio = BIO_new_connect("hostname:port");

if(bio == NULL)
{
   /* trateaza eroarea */
}
if(BIO_do_connect(bio) <= 0)
{
   /* trateaza conexiune esuata*/
}
```

Citirea/scrierea de la/la un obiect BIO se face cu functiile *BIO_read()* si respectiv *BIO_write()*.

```
int BIO_read(BIO *b, void *buf, int len);
int BIO_write(BIO *b, const void *buf, int len);
```

Functia *BIO_read()* incearca sa citeasca len octeti din b si sa plaseze rezultatul in bufferul buf. Pentru o conexiune blocanta 0 inseamna ca conexiunea a fost inchisa in timp ce -1 indica ca a aparut o eroare. Pentru o conexiune neblocanta o valoare 0 inseamna ca nu s-a citit nimic iar -1 ca a aparut o eroare. In acest caz se poate reincerca operatia care a dat eroare, cu *BIO_should_retry()*.

```
int x = BIO_read(bio, buf, len);
if(x == 0)
{
   /* trateaza conexiune inchisa*/
}
else if(x < 0)
{
   if(! BIO_should_retry(bio))
   {
      /* trateaza esec operatie read */
   }
   /* trateaza operatie reusita de retry */
}
```

Functia *BIO_write()* incearca sa scrie len octeti din buf la b.In ambele variante o valoare returnata de -2 inseamna ca functia nu e implementata pentru tipul specific BIO folosit.

```
if(BIO_write(bio, buf, len) <= 0)
{
if(! BIO_should_retry(bio))
{
   /* trateaza esec operatie write */
}
   /* trateaza retry reusit */
}
```

Conexiunea poate fi inchisa in doua moduri, cu *BIO_reset()* astfel incat poate fi reutilizata ulterior sau cu *BIO_free_all()* caz in care se elibereaza memoria alocata si se inchide socketul asociat.

```
/* reutilizare conexiune*/
BIO_reset(bio);
/* eliberare memorie */
BIO_free_all(bio);
```

Stabilirea unei conexiuni securizate

Conexiunile securizate necesita un handshake dupa stabilirea conexiunii. In cursul handshake-ului, serverul trimite clientului un certificat, pe care apoi clientul il verifica folosind un set de certificate (certificate store) pe care le considera de incredere (trusted). Deasemenea verifica daca certificatul nu a expirat. Clientul trimite un certificat catre server numai daca acesta din urma cere unul (autentificarea clientului). Folosind transferul de certificate se face setup-ul conexiunii securizate. Clientul sau serverul pot solicita un nou handshake la orice moment. RFC 2246 detaliaza aspectele referitoare la protocolul de handshake.

Pentru a stabili o conexiune securizata mai sunt necesari doi pointeri unul de tip *SSL_CTX* (context object) si unul de tip SSL.

```
SSL_CTX * ctx = SSL_CTX_new(SSLv23_client_method());
SSL * ssl;
```

Urmatorul pas este includerea certificate store mentionat anterior. OpenSSL furnizeaza un set de certificate trusted. In arhiva cu codul sursa din distributia OpenSSL se afla un fisier "TrustStore.pem." care include toate certificatele trusted.

Functia *SSL_CTX_load_verify_ locations()* e folosita pentru a incarca acest fisier. Aceasta functie are trei parametri: pointerul la context calea catre fisierul *.pem si calea catre un director ce contine certificate. Unul din ultimii doi parametri trebuie specificat (ori fisierul *.pem sau, alternativ, directorul ce contine certificatele trusted). Returneaza 1 in caz de succes si 0 daca a aparut o problema.

```
if(! SSL_CTX_load_verify_locations(ctx, "TrustStore.pem", NULL))
{
   /* trateaza esec incarcare trust store */
}
```

Crearea unei noi conexiuni securizate se face cu:

```
bio = BIO_new_ssl_connect(ctx);
BIO_get_ssl(bio, & ssl);
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);
// cu aceasta optiune daca serverul
// doreste sa initieze din nou un handshake la un moment oarecare de timp,
// OpenSSL se va ocupa in background de acest task
```

Dupa ce acest set-up a fost realizat se poate trece la deschiderea propriu-zisa a conexiunii

```
/* incearca sa se conecteze */
BIO_set_conn_hostname(bio, "hostname:port");

/* verifica conexiunea deschisa si efectueaza handshake */
if(BIO_do_connect(bio) <= 0)
{
   /* trateaza esec conexiune */
}
```

Pentru a verifica daca validarea certificatului s-a facut cu succes se apeleaza *SSL_get_verify_result()* cu unic parametru structura SSL. Daca validarea s-a efectuat cu succes returneaza X509 V OK, altfel intoarce un cod de eroare care poate fi documentat daca se foloseste optiunea verify in linia de comanda a utilitarului.

```
if(SSL_get_verify_result(ssl) != X509_V_OK)
{
   /* trateaza esec verificare */
}
```

Stiva de erori poate fi scrisa intr-un fisier cu *ERR_print_errors_fp(FILE *);* Erorile au urmatorul format:

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

Cerinte laborator

Completati scheletul de cod astfel incat sa se efectueze o conexiune prin HTTPS la verisign.com si sa se salveze local, intr-un fisier pe disc, pagina home, printr-un apel de tip GET.

Portul HTTPS implicit este 443.

Pentru compilare se poate folosi:

```
gcc http_client.c -lssl -lcrypto
```

Referinte

1. http://www.openssl.org/
2. http://www.linuxjournal.com/article/4822
3. http://www.rfc-editor.org/rfc/rfc2246.txt

Cursuri

- Cursul 01.
- Cursul 02.
- Cursul 03.
- Cursul 04.
- Cursul 05.
- Cursul 06.
- Cursul 07.
- Cursul 08.
- Cursul 09.
- Cursul 10.
- Cursul 11.
- Cursul 12.

Laboratoare

- Laboratorul 01 - Notiuni pregatitoare pentru laboratorul de PC
- Laboratorul 02 - Folosirea unei legaturi de date pentru transmiterea unui fisier
- Laboratorul 03 - Implementarea unui protocol cu fereastra glisanta. Suma de control
- Laboratorul 04 - Forwarding
- Laboratorul 05 - ICMP
- Laboratorul 06 - Socketi UDP
- Laboratorul 07 - Protocolul de transport TCP
- Laboratorul 08 - TCP și multiplexare I/O
- Laboratorul 09 - Protocolul DNS
- Laboratorul 10 - Protocolul HTTP
- Laboratorul 11 - E-mail
- Laboratorul 12 - Protocoale de securitate. OpenSSL CLI tools
- Laboratorul 13 - Protocoale de securitate. utilizarea programatica

Resurse

- Mașina virtuală

Table of Contents