Conexiune Securizată cu OpenSSL

Introducere

OpenSSL este cea mai cunoscută librărie criptografică gratuită pentru comunicații sigure, iar o simplă căutare pe Google referitoare la acest subiect ne confirmă spusele. A fost creată în 1998, având la bază librăria *SSLeay*. Dintre celelalte biblioteci SSL putem aminti *GnuTLS* și *Network Security Services (NSS)*.

Cu toate acestea, de ce preferăm OpenSSL în detrimentul competiției? Licențierea este unul dintre aspectele importante. Mozilla NSS este distribuită atât sub licență publică Mozilla, cât și sub GnuGPL, permițând dezvoltatorului să aleagă. Dar NSS este mai mare decât OpenSSL și necesită alte biblioteci externe pentru a o putea rula, în timp ce OpenSSL este complet autonomă. În plus, OpenSSL folosește ultimele versiuni de protocoale SSL/TLS, spre deosebire de GnuTLS.

Ce înseamnă SSL?

SSL este un acronim care vine de la *Secure Sockets Layer*. Este standardul din spatele comunicării securizate pe Internet. Certificatele și algoritmii criptografici stau la baza modului în care funcționează și, cu OpenSSL, avem ocazia să ne jucăm cu ambele. Protocolul SSL oferă diverse servicii de securitate, inclusiv două care sunt centrale în **HTTPS**:

- *Autentificare de la egal la egal*: fiecare parte a unei conexiuni autentifică identitatea celeilalte părți. Dacă Alice și Bob vor schimba mesaje prin SSL, atunci fiecare autentifică mai întâi identitatea celuilalt.
- *Confidențialitate*: un expeditor criptează mesajele înainte de a le trimite pe un canal de comunicare. Receptorul decriptează apoi mesajul primit. Acest proces protejează conversațiile din rețea. Chiar dacă ascultătorul, Eve, interceptează un mesaj criptat de la Alice către Bob, se consideră că Eve nu poate să descifreze acest mesaj.

La rândul lor, aceste două servicii cheie sunt legate de altele care primesc mai puțină atenție. De exemplu, SSL acceptă *integritatea mesajului*, ceea ce asigură că un mesaj primit este același cu cel trimis. Această caracteristică este implementată cu *funcții hash*, care vin, de asemenea, în pachetul de instrumente OpenSSL.

Conexiunile SSL și securizate pot fi utilizate pentru orice tip de protocol de pe Internet, indiferent dacă este vorba de **HTTP**, **POP3** sau **FTP**. Nu este necesar să folosim SSL pentru orice tip de conexiune. Ar trebui utilizat dacă conexiunea va conține informații sensibile (**parole, date personale**, etc...).

Ce este OpenSSL?

OpenSSL este o bibliotecă foarte complexă, pe care nu o putem prezenta complet doar în câteva rânduri. Pe scurt, ea este capabilă să cripteze și să decripteze fișiere, poate integra certificate digitale, semnături digitale etc... OpenSSL este mai mult decât un API, este și un instrument de linie de comandă. Instrumentul pentru linia de comandă poate face aceleași lucruri ca API-ul, dar duce lucrurile un pas mai departe, permițând posibilitatea de a testa

serverele și clienții SSL. Deci, putem utiliza OpenSSL pentru dezvoltarea unui proiect complex și astăzi voi discuta despre modul în care se implementează **o conexiune securizată**, de bază.

Configurarea unei Conexiuni Nesecurizate

Vom începe cu prezentarea configurării unei conexiuni nesigure, pentru a înțelege pașii de bază, apoi vom adăuga elementele care ne vor ajuta să o transformăm într-una securizată.

OpenSSL folosește o *bibliotecă de abstractizare* numită **BIO** pentru a gestiona comunicarea de diferite tipuri, inclusiv fișiere și socket-uri, fie ele sigure, sau nu. O conexiune standard, se realizează astfel:

```
/* OpenSSL headers */
#include <openssl/bio.h>   // BasicInput/Output streams
#include <openssl/err.h>   // errors
#include <openssl/ssl.h>   // core library

/* Initializing OpenSSL */
SSL_load_error_strings(); /* registers the error strings for all libcrypto
functions, but also registers the libssl error strings */
SSL_library_init(); /* registers the available SSL/TLS ciphers and digests -
OpenSSL_add_ssl_algorithms() and SSLeay_add_ssl_algorithms() are synonyms for
SSL_library_init() */

/* Creating and opening a connection */
BIO *bio = BIO_new_connect("hostname:port");
if(bio == NULL)
{ /* Handle the failure */ }
if(BIO_do_connect(bio) <= 0)
{ /* Handle failed connection */ }
```

Crearea unei noi conexiuni necesită un apel către **BIO_new_connect**. Odată ce *hostname-ul* și *numărul portului* sunt specificate, obiectul BIO va încerca să deschidă conexiunea. Dacă a apărut o problemă la crearea obiectului, pointerul va fi **NULL**. Efectuăm un apel către **BIO_do_connect** pentru a verifica dacă conexiunea a avut succes (**0** sau **-1** la eroare). De exemplu, dacă vrem să ne conectăm prin *portul 80* la *www.google.com*, string-ul citit ar fi ***www.google.com:80***.

BIO_read va încerca să citească un anumit număr de octeți de pe server. Returnează **numărul de octeți citiți**, **0** sau **-1**. La o conexiune pe modul *block*, **0** înseamnă **conexiune închisă**, în timp ce **-1** indică faptul că a apărut o **eroare**. La o conexiune care **nu** este în acest mod, **0** înseamnă că **nu există date disponibile**, iar **-1** indică o **eroare**. Pentru a determina dacă eroarea este recuperabilă, trebuie apelat **BIO_should_retry**.

```
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 */
}
```

Procedeul este similar și pentru **BIO_write**.

```
if(BIO_write(bio, buf, len) <= 0)
{
    if(!BIO_should_retry(bio))
    { /* Handle failed write here */ }

    /* Do something to handle the retry */
}
```

Pentru a închide conexiunea folosim **BIO_reset** sau **BIO_free_all**. **BIO_reset** o închide și resetează starea internă a obiectului BIO, astfel încât conexiunea să poată fi reutilizată. Acest lucru este bun dacă dorim să utilizăm același obiect în întreaga aplicație, cum ar fi cu un client dintr-un chat sigur. **BIO_free_all** șterge structura internă și toată memoria asociată, și închide socket-ul aferent.

```
// To reuse the connection, use this line
BIO_reset(bio);

// To free it from memory, use this line
BIO_free_all(bio);
```

Configurarea unei Conexiuni Securizate

Conexiunile securizate necesită un *handshake* după stabilirea conexiunii dintre server și client. În timpul acestui *handshake*, serverul trimite un certificat către client, pe care clientul îl verifică în raport cu un set de certificate de încredere și, în același timp, se asigură că acesta nu a expirat. Clientul va trimite un certificat serverului numai dacă serverul solicită unul (autentificare client). Folosind certificatul (certificatele), parametrii de criptare sunt transferați între client și server pentru a configura conexiunea securizată. Chiar dacă se efectuează un *hanshake* după stabilirea conexiunii, clientul sau serverul poate solicita unul nou în orice moment.

Acum este timpul să adăugăm elementele care transformă o conexiune nesigură într-una sigură:

```
SSL_CTX *ctx = SSL_CTX_new(SSLv23_client_method()); /* creates a new SSL_CTX object
as framework to establish TLS/SSL enabled connections */
SSL *ssl;
```

Următorul pas este să încărcăm *a trust certificate store*. Acest lucru este vital pentru funcționarea conexiunii. Certificatele de încredere din OpenSSL sunt incluse în arhiva codului sursă într-un singur fișier numit **TrustStore.pem**.

```
// load trust certificate store for the upcoming verification
if(!SSL_CTX_load_verify_locations(ctx, "/path/to/TrustStore.pem", NULL))
{ perror("SSL_CTX_load_verify_locations..."); }
```

Acum ne vom ocupa de crearea conexiunii. Cu opțiunea **SSL_MODE_AUTO_RETRY** setată, dacă serverul dorește brusc un nou *handshake*, OpenSSL o gestionează în fundal. Fără această opțiune, orice operație de citire sau scriere va returna o eroare în cazul în care serverul dorește un nou *handshake*.

```
/* Opening a secure connection */
bio = BIO_new_ssl_connect(ctx);
BIO_get_ssl(bio, &ssl); // session
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY); // robustness
BIO_set_conn_hostname(bio, "hostname:port");  // prepare to connect
```

```
// try to connect
if(BIO_do_connect(bio) <= 0)
{
  cleanup(ctx, bio);
  perror("BIO_do_connect...");
}
```

Odată ce conexiunea este stabilită, certificatul trebuie verificat pentru a vedea dacă este valid. OpenSSL face acest lucru pentru noi. Dacă există probleme fatale cu certificatul (de exemplu, dacă valorile hash nu sunt valide) atunci conexiunea pur și simplu nu se va întâmpla. Dar dacă există probleme non-fatale cu certificatul (ca atunci când acesta a expirat sau nu este încă valid) conexiunea poate fi utilizată în continuare.

```
// check if a certificate is valid
if(SSL_get_verify_result(ssl) != X509_V_OK)
{ /* Handle the failed verification */ }
```

La un moment dat înainte de sfârșitul aplicației, structura de context SSL trebuie eliberată: `SSL_CTX_free(ctx);`

Concluzie

Crearea unei conexiuni sigure, de bază, cu OpenSSL nu este dificilă, dar documentația poate fi puțin intimidantă atunci când încercăm să facem acest lucru. Acest proiect a încercat să prezinte o introducere referitoare la acest subiect, dar există destul de multă flexibilitate cu OpenSSL și setări avansate (care nu au fost abordate în aceste pagini) de care este posibil să avem nevoie pentru a implementa în mod adecvat funcționalitatea SSL, în cadrul unui proiect mai avansat. Cu toate acestea, codul prezentat mai sus reprezintă un bun punct de plecare.

Bibliografie:

1. Wikipedia Contributors. OpenSSL. Wikipedia. Publicat pe 7 Aprilie, 2021. Accesat pe 11 Aprilie, 2021. https://en.wikipedia.org/wiki/OpenSSL

2. OpenSSL Foundation, Inc. /index.html. Openssl.org. Publicat în 2021. Accesat pe 11 Aprilie, 2021. https://www.openssl.org/

3. Wikipedia Contributors. Comparison of TLS implementations. Wikipedia. Publicat pe 19 Februarie, 2021. Accesat pe 11 Aprilie, 2021. https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations

4. NSS FAQ - Mozilla | MDN. Mozilla.org. Publicat pe 9 Martie, 2021. Accesat pe 11 Aprilie, 2021. https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/FAQ

5. Wikipedia Contributors. GnuTLS. Wikipedia. Publicat pe 11 Martie, 2021. Accesat pe 11 Aprilie, 2021. https://en.wikipedia.org/wiki/GnuTLS

6. Wikipedia Contributors. Network Security Services. Wikipedia. Publicat pe 13 Martie, 2021. Accesat pe 11 Aprilie, 2021. https://en.wikipedia.org/wiki/Network_Security_Services

7. steve. SSL and SSL Certificates Explained For Beginners. Publicat pe 27 Octombrie, 2016. Accesat pe 11 Aprilie, 2021. http://www.steves-internet-guide.com/ssl-certificates-explained/

8. Wikipedia Contributors. Transport Layer Security. Wikipedia. Publicat pe 11 Aprilie, 2021. Accesat pe 11 Aprilie, 2021. https://en.wikipedia.org/wiki/Transport_Layer_Security

9. Getting started with OpenSSL: Cryptography basics. Opensource.com. Publicat în 2019. Accesat pe 11 Aprilie, 2021. https://opensource.com/article/19/6/cryptography-basics-openssl-part-1

10. An Introduction to OpenSSL Programming, Part I of II | Linux Journal. Linuxjournal.com. Publicat în 2021. Accesat pe 11 Aprilie, 2021. https://www.linuxjournal.com/article/4822

11. Rfc-editor.org. Publicat în 2021. Accesat pe 11 Aprilie, 2021. https://www.rfc-editor.org/rfc/rfc2246.txt