Programmation Réseau

SSH et TLS (aka SSL)

Jean-Baptiste.Yunes@liafa.jussieu.fr

Coloriages: François Armand
armand@informatique.univ-paris-diderot.fr

UFR Informatique

2011-2012

Problèmes principaux:

Authentification

Confidentialité

Intégrité

Authentification:

prétend »

Éviter les imposteurs, attaques de type « man in the middle »

Confidentialité:

Ne pas dévoiler des informations sans autorisation

Donc empêcher

Utilisateur non autorisé d'accéder à une information à laquelle il ne devrait pas avoir accès,

Utilisateur autorisé de transmettre une information à un tiers non autorisé

Intégrité:

Absence d'altérations incorrectes de l'état du système

Empêcher les modifications illégitimes

Par des utilisateurs non autorisés

Ou par des utilisateurs autorisés

Empêcher qu'on puisse empêcher les modifications légitimes

Du point de vue des communications, la confidentialité se base sur le cryptage / chiffrement des informations

Cryptage avant envoi

Transfert

Décryptage à la réception

Problème: comment crypter / décrypter de manière cohérente

Asymétrique:

Chaque « partie » une clé (privée/publique)

Un message codé avec une des clés ne peut-

$MsgCrypté = f_{crypt}(Msg, clé_{publique})$

$Msg = f_{decrypt}(MsgCrypté, clé_{privée})$

Ou

$MsgCrypté = f_{crypt}(Msg, clé_{privée})$

$Msg = f_{decrypt}(MsgCrypté, clé_{publique})$

Problème:

Les mécanismes asymétriques sont beaucoup plus « onéreux » que les mécanismes symétriques.

Idée:

Utiliser le chiffrement asymétrique pour transmettre un clé de chiffrement symétrique

Communications entre machines

Remplacées par:

ssh

scp

sftp

ssh:

démon » (sshd) sur la machine à laquelle on veut se connecter

port 22 par défaut

Syntaxe:

ssh user@hostname

Nombreuses options, port forwarding, X11 forward

- serveur
  - Le serveur génère une paire de clés (chiffrage asymétrique)
  - Envoi de la clé publique au client
- Le client génère une clé de chiffrage symétrique
  - Envoi de la clé symétrique cryptée avec la clé publique reçue.
- Échanges
  - Confidentialité: cryptés par la clé « symétrique »

(optionnel)

Un utilisateur génère une paire de clés sur sa machine

ssh-keygen t dsa

Ou rsa (ssh V1)

ssh

Transfert de la clé publique sur le serveur auquel on voudra se connecter:

ssh-copyid user@hostname

ssh

Plus besoin de rentrer un mot de passe

On peut protéger le stockage des cles par une passphrasse

rsync permet de « copier » une arborescence de

La commande peut-elle-même être exécutée depuis A, B ou même C!

Transfert « intelligent »

Transfert « réduit » aux modifications depuis la dernière copie (via rsync).

Exemple:

rsync toto francois@lucien:/home/francois/exo

rsync peut utiliser

ssh comme couche de transport

rsync --rsh=ssh toto francois@lucien**:**/tmp/exo

TCP en se connectant à un démon rsync

rsync toto francois@lucien**::**/tmp/exo

TLS (aka SSL)

Transport Layer Security (aka Secure Socket Layer) est une couche intermédiaire entre les couches Transport et Application de la pile Internet et joue un rôle équivalent à la couche Session du modèle OSI

elle fournit des services de sécurité suivant :

authentification

confidentialité

Intégrité

Protocoles similaires mais différents

intéressantes :

- la transparence (les protocoles applicatifs ne sont pas modifiés)

- certain niveau de sécurité, i.e. pas de configuration a-priori)

la première normalisation correspondante est la RFC 2246 (01/1999)

- il en existe une version « paquet », DTLS normalisée dans RFC 4347

asymétrique (type RSA) pour établir une liaison sûre utilisant de la cryptographie symétrique (type DES)

- la cryptographie asymétrique est utilisée pour échanger la clef de cryptage de la communication

En général une authentification est aussi réalisée

pour le serveur il faut une paire de clefs cryptographiques (une privée, une publique)

ces clefs seront stockées dans une armoire à clefs (keystore), cette armoire étant elle-même protégée par un mot de passe

keytool -genkey

-alias nomclef

-keyalg RSA

-keystore armoire.jks

[-storepass mpd]

pour le client il faut une armoire de clefs contenant le certificat correspondant à la clef publique du serveur

keytool -export

-alias nomclef

-keystore armoire.jks

[-storepass mdp]

-file certificat.cert

tout en le reconnaissant comme certificat acceptable

keytool -importcert

-trustcacerts

-alias nomclef

-file certificat.cer

-keystore armoireclient.jks

[-storepass mdp]

Côté serveur la création de la socket

```
ServerSocketFactory ssocketFactory =

 SSLServerSocketFactory.getDefault();

ServerSocket ssocket =

 ssocketFactory.createServerSocket(port);
```

```
par

SocketFactory socketFactory =

 SSLSocketFactory.getDefault();

Socket socket =

 socketFactory.createSocket(hostname,port);
```

Attention les instructions précédentes ne

la spontanéité

authentification)

Elles ne doivent pas être utilisées pour établir une liaison « commerciale »

le certificat est auto-
point de vue de la sécurité et sensibilité à des attaques)

Exemple: serveur

```
SSLServerSocketFactory.getDefault();

ssocketFactory.createServerSocket(port);
```

Exemple : client

```
SSLSocketFactory.getDefault();
socketFactory.createSocket(hostname, port);
```

Exemple : keystores

Dans le répertoire du serveur:

keytool -genkey -keystore demo.ks

keytool -export file demo.cer -keystore demo.ks

Dans le répertoire du client

keytool -import -file demo.cer -keystore clientdemo.ks

Exemple : exécution

Côté serveur:

```
java -Djavax.net.ssl.keyStore=demo.ks
-Djavax.net.ssl.keyStorePassword=123456
    Serveur
```

Côté client:

```
java -Djavax.net.ssl.trustStore= clientdemo.ks
-Djavax/net/ssl/trustStorePassword=123456
    Client
```

Client sans utilisation de keystore

```
SSLContext.getInstance("TLS");
SavingTrustManager();
context.init(null, new TrustManager[]{tm}, null);
context.getSocketFactory();
```

Client sans utilisation de keystore

Il existe différentes implémentations

OpenSSL http://*www.**openssl**.org/*

1.0.1 du 14/03/2012

Supporte les différentes versions du protocole (ssl v1, v2, v3, tls v1)

Crypto: AES, Blowfish, Camellia, SEED, CAST-128, DES, IDEA, RC2, RC4, RC5, Triple DES, GOST 28147-89

Hash: MD5, MD2, SHA-1, SHA-2, RIPEMD-160, MDC-2, GOST R 34.11-94

Public key: RSA, DSA, Diffie Hellman key exchange, Elliptic curve, GOST R 34.10-2001

-tutorial-client-t11.html

université
PARIS
DIDEROT
PARIS 7

université
PARIS
DIDEROT
PARIS 7

SSL_CTX_new

BIO_new_ssl_connect

BIO_get_ssl

SSL_set_mode

BIO_set_conn_hostname "https"

BIO_do_connect(bio)

BIO_do_handshake(bio)