Documentation de Stage de 1ère Année

ES-SAIDI Naoufal

Sommaire

I - Installation d'OMNeT++
- Installation et configuration
- Dépendances
- Compilation
- Problèmes rencontrés

II - Installation de NS-3
- Dépendances
- Installation et compilation
- Exécution

III - Présentation du projet
- Objectif
- Principe Diffie-Hellman

IV - Architecture du réseau (NED)
- Topologie
- Rôle des nœuds

V - Configuration (omnetpp.ini)

VI - Implémentation (.cc)
- Échange de clés
- Interception

VII - Résultats et limites
VIII - Conclusion

I - Installation d'OMNeT++ sur Debian 12

OMNeT++ est un framework de simulation en C++ extensible et modulaire, fondé sur une architecture à base de composants. Il est principalement utilisé pour développer des simulateurs de réseaux.

Étape 1 : Installation et Configuration de Debian 12 sur Oracle VirtualBox

Avant de commencer l'installation **d'OMNeT++,** il est recommandé de s'assurer que le système est à jour :

```
sudo apt update && sudo apt upgrade -y
```

Étape 2 : Installer les dépendances requises

OMNeT++ nécessite plusieurs outils et bibliothèques pour être compilé :

```
sudo apt install build-essential g++ python3 python3-pip python3-venv qt5-qmake qtbase5-dev qtchooser libqt5svg5-dev libxml2-dev libsqlite3-dev libboost-all-dev libssl-dev libcurl4-openssl-dev libsdl1.2-dev libfftw3-dev libosg-dev -y
```

Problèmes rencontrés :

- **Python3 et pip** :** Sur Debian, python3 et pip sont installés par défaut, mais si pip ou python3 ne sont pas présents, il faut les installer.
- **Qt**** : Il faut absolument avoir Qt 5.9 ou plus récent pour la partie graphique de OMNeT++.
- **OSG (OpenSceneGraph)**** : Désactiver OSG pour ne pas avoir de vue 3D si non nécessaire.

Étape 3 : Télécharger OMNeT++

Téléchargement du fichier précompilé omnetpp-6.1.0-linux-x86_64.tgz puis extraction :

```bash
tar -xzvf omnetpp-6.1.0-linux-x86_64.tgz
```

Étape 4 : Configurer l'environnement avec setenv

Le script setenv permet de configurer l'environnement pour pouvoir utiliser OMNeT++ sans problème.

Étape 5 : Lancer le script ./configure

Problèmes rencontrés :

- **Erreur Makefile.inc manquant** : `config.status: error: Cannot find input file: 'Makefile.inc.in'`

- **Erreur avec Qt** : `configure error: Cannot build Qt apps`
- **Erreur liée à OpenSceneGraph**

Solutions :

- Ré-extraire le fichier omnetpp-6.1.0-linux-x86_64.tgz
- Installer Qt 5.9+ via `sudo apt install qt5-qmake qtbase5-dev`
- Désactiver Qt et OSG dans configure.user : `WITH_QTENV=no` et `WITH_OSG=no`

Étape 6 : Compiler OMNeT++

```
make distclean
./configure
make -j$(nproc)
```

Lancement de l'IDE OMNeT++ réussi !

II - Installation de NS-3 sur Debian 12

NS-3 (Network Simulator 3) est un simulateur open-source d'événements discrets, conçu pour modéliser et étudier le comportement des réseaux Internet. Destiné à la recherche et à l'enseignement, il est développé collaborativement sous licence GNU GPLv2.

Étape 1 : Installer les dépendances requises

```
sudo apt update
sudo apt install -y git g++ python3 python3-dev python3-pip cmake ninja-build pkg-config libsqlite3-dev libxml2-dev qtbase5-dev qt5-qmake qtbase5-dev-tools libgtk-3-dev libboost-all-dev libgsl-dev libssl-dev libpcap-dev libasio-dev liblz4-dev libopenmpi-dev openmpi-bin
```

Étape 2 : Télécharger ns-3.36

```
wget https://www.nsnam.org/release/ns-allinone-3.36.tar.bz2
tar -xjf ns-allinone-3.36.tar.bz2
```

Étape 3 : Compiler et configurer ns-3.36 avec CMake

```
./build.py --enable-examples --enable-tests
```

Étape 4 : Exécuter des simulations

```
./ns3 run examples/tutorial/first
./ns3 run scratch/BTSSIO
```

```

1. **Présentation générale du projet :**

Le projet consiste à simuler un échange sécurisé entre deux nœuds A et B en utilisant le protocole de **Diffie-Hellman** key exchange, puis à chiffrer les communications à l'aide d'un chiffrement symétrique inspiré de **Advanced Encryption Standard.**

Le projet est structuré **en trois types de fichiers :**

- fichiers .**ned** → description du réseau,
- fichier .**ini** → configuration de la simulation,
- fichiers .**cc** → logique des modules.

2. **Fichier NED : Description du réseau**

Le fichier .ned définit la topologie du réseau, c'est-à-dire :

- les modules (A, B, E),
- les connexions entre eux.

- **A et B** sont les nœuds communicants.
- **E** est l'attaquant passif.
- Tous les messages passent par E → simulation d'interception.

Node.cc | omnetpp.ini | DiffieHellman.ned ×

```
simple Node {
    parameters:
        string nodeName;
    gates:
        input in;
        output out;
}

network DiffieHellmanNetwork {
    submodules:
        A: Node { parameters: nodeName = "A"; }
        B: Node { parameters: nodeName = "B"; }
        E: Node { parameters: nodeName = "E"; }

    connections:
        A.out --> E.in;
        E.out --> B.in;

        B.out --> E.in;
        E.out --> A.in;
}
```

- **Connexions entre les nœuds :**

**Communication A → B**

- A envoie → E intercepte → B reçoit
  A → E → B

**Communication B → A**

- B envoie → E intercepte → A reçoit
  B → E → A
```

Analyse de la topologie

Cette architecture impose que **tous les messages passent par E**.

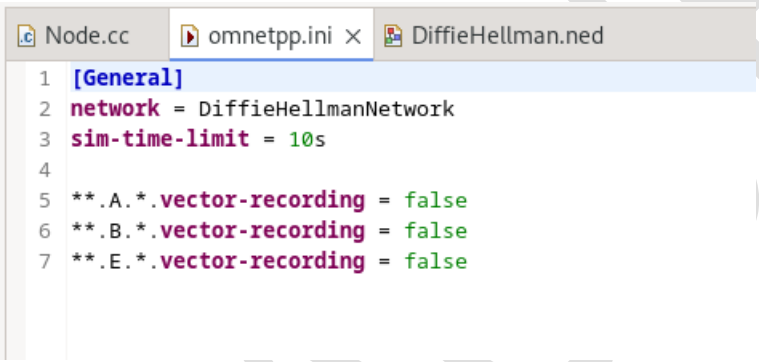
Conséquences :

- E peut :
  - intercepter les messages,
  - observer les échanges (clés publiques, messages chiffrés).
- E ne peut pas :
  - modifier les messages (attaquant passif),
  - calculer la clé secrète.

3. Fichier omnetpp.ini : Configuration

Ce fichier **configure la simulation** :

- réseau utilisé,
- durée de simulation,
- paramètres globaux.



vector-recording :
Ce paramètre permet de contrôler l'enregistrement des données vectorielles pendant la simulation.
Les vecteurs servent à stocker :

- l'évolution des valeurs dans le temps,
- les statistiques (délais, trafic, etc.).

4. Fichier .cc

Node.cc × omnetpp.ini DiffieHellman.ned

```
#include <omnetpp.h>
#include <cmath>
#include <string>

using namespace omnetpp;

class Node : public cSimpleModule {
  private:
    std::string name;
    double g = 5.0;         // base commune
    double privateKey;      // a ou b
    double publicKey;       // g^a ou g^b
    double receivedKey;     // clé reçue de l'autre partie
    double sharedSecret;    // clé partagée

  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

```

```
Define_Module(Node);

void Node::initialize() {
    name = par("nodeName").stdstringValue();

    if (name == "A" || name == "B") {
        privateKey = name == "A" ? 3 : 4;  // A a=3, B b=4
        publicKey = pow(g, privateKey);

        // Envoyer publicKey à l'autre via E
        cMessage *msg = new cMessage("PublicKey");
        msg->setKind(0);
        msg->addPar("sender") = name.c_str();
        msg->addPar("value") = publicKey;
        send(msg, "out");
    }
}

```

Cette fonction est exécutée au démarrage de la simulation.

Étapes :

1. **Récupération du nom du nœud :**

name = par("nodeName").stdstringValue();

2. **Attribution de la clé privée :**

privateKey = (name == "A") ? 3 : 4;

3. **Calcul de la clé publique :**

publicKey = pow(g, privateKey);

4. **Envoi de la clé publique :**

```
cMessage *msg = new cMessage("PublicKey");
msg->addPar("sender") = name.c_str();
msg->addPar("value") = publicKey;
send(msg, "out");
```

```
37 }
38
39 void Node::handleMessage(cMessage *msg) {
40     std::string sender = msg->par("sender").stdstringValue();
41     double otherKey = msg->par("value").doubleValue();
42
43     if (name == "E") {
44         EV << "E intercepte un message de " << sender
45            << " avec valeur = " << otherKey << "\n";
46         // E retransmet sans modifier
47         cMessage *forward = msg->dup();
48         send(forward, "out");
49         delete msg;
50     }
51     else {
52         EV << name << " reçoit la clé publique de " << sender
53            << " = " << otherKey << "\n";
54         receivedKey = otherKey;
55         sharedSecret = pow(receivedKey, privateKey);
56         EV << name << " calcule la clé secrète partagée = "
57            << sharedSecret << "\n";
58         delete msg;
```

Cas 1 : Nœud E (intercepteur)

```
if (name == "E") {
   EV << "E intercepte un message de " << sender;

   cMessage *forward = msg->dup();
   send(forward, "out");
   delete msg;
}
```

Le nœud E :

- affiche le contenu
- retransmet sans modification

Cas 2 : Nœuds A ou B

```
receivedKey = otherKey;
sharedSecret = pow(receivedKey, privateKey);
```

Chaque nœud :

1. reçoit la **clé publique** de l'autre
2. calcule la **clé secrète partagée**

Exemple de fonctionnement

Avec :

- g = 5
- A = 3
- B = 4

Calculs :

- A → $5^3 = 125$
- B → $5^4 = 625$

Clé finale :

- A calcule : 625^3
- B calcule : 125^4

Résultat identique → clé partagée

Cette simulation est simplifiée :

- *pas de modulo (contrairement au vrai protocole)*
- *utilisation de double (non sécurisé)*
- *vulnérable à une attaque de type **Man-in-the-Middle***

Le nœud E montre justement cette faiblesse.

Conclusion :

Ce projet a permis de simuler le fonctionnement du protocole Diffie-Hellman dans un environnement réseau avec OMNeT++. Il montre comment deux entités peuvent générer une clé secrète commune sans la transmettre directement, tout en mettant en évidence les limites de sécurité, notamment face à une interception. Cette simulation constitue une bonne introduction aux bases de la cryptographie et de la sécurité des échanges réseau.