Projet : Réseau

Dans ce projet, nous allons nous intéresser à plusieurs aspects de la partie réseau du cours [1].

Le projet est à faire par groupes de 3. Le rendu des codes source s'effectuera le **11 mai 2018** par courrier électronique *aux trois TDmen* [2] avant **20h**, dans le format suivant : une archive `nom1-nom2-nom3-Projet.tgz`, contenant un répertoire `nom1-nom2-nom3-Projet/` qui lui-même contiendra les fichiers demandés. L'objet du mail devra être `[ASR2] Rendu projet.` [3]

De plus, nous vous demandons un rapport *individuel* **par personne**. Il est à rendre pour la même date, au format pdf, et intitulé `Rapport-nom.pdf`, aux trois TDmen.

Vous pouvez bien évidemment nous poser des questions, (de vive voix ou par mail aux **trois** chargés de TP), si vous bloquez sur une question.

Exercice 1. *Malware*

Joseph Marchand, chercheur à l'ENS de Lyon, a vécu récemment une bien mauvaise aventure : il a été infecté par un malware ! Il était connecté au VPN de l'ENS de Lyon. Faisant partie de la DSI, vous avez pu enregister son trafic internet [4]. Nous pensons que le vecteur d'infection a été un **email** qui comportait un lien malicieux sur lequel il a cliqué par mégarde. Il ne vous reste plus qu'à enquêter pour parvenir à la source de l'infection, les mouflons connectés. Vous pourrez utiliser pour ces questions un logiciel bien pratique : wireshark.

1. Joseph a affirmé qu'il était à son bureau à ce moment-là. Est-ce vrai ? Donnez une preuve.
2. À l'aide de wireshark, en regardant les requêtes DNS faites par Joseph, retracez son parcours sur le web. Quel filtre wireshark avez-vous utilisé ? Combien de requêtes DNS ont été faites ?
3. En déduire où Joseph a téléchargé le malware. Quel est le lien de téléchargement ?
4. À partir du malware, trouvez à quel serveur le malware tente de se connecter.
5. Repérez la requête DNS correspondant à ce serveur. Quelle est son adresse IP ?
6. Sur quel port le malware communique-t-il avec le serveur ? Comment l'avez-vous trouvé ?
7. Le malware envoie des identifiants (login et mot de passe) au serveur. À l'aide de wireshark, trouvez-les. Indiquez-nous comment vous avez procédé pour les trouver.
8. Que fait le malware ? Dans le dump fourni, a-t-il envoyé d'autres informations que les identifiants ?
9. Lancez-le sur votre machine [5], et observez son trafic avec wireshark. Quelle information n'envoie-t-il pas ?
10. Expliquez pourquoi le malware ne se comporte pas de la même façon. À l'aide de gdb, essayez de le faire fonctionner complètement. Indiquez comment vous avez fait.
11. (bonus) Il est possible de trouver les identifiants à partir du malware (c'est aussi un crackme !). Si vous y arrivez, indiquez-nous comment vous avez fait.

Exercice 2. *Un client haut en couleurs*

1. avec un chouilla de système quand même
2. `{remy.grunblatt,valentin.lorentz,alexandre.talon}@ens-lyon.fr`
3. le point terminait la phrase, et ne doit pas être présent dans l'objet du mail
4. il est à cette adresse : `https://remy.grunblatt.org/teaching/ASR2/Projet_dump.pcap`
5. Rassurez-vous, il ne fait rien de méchant.

Ici, le but est de communiquer avec un serveur... le tout avec un programme C. Nous utiliserons ici le protocole TCP. Nous avons donc mis en place un serveur, qui répondra à l'adresse 62.210.74.95, port 11111.

Après avoir initié la connexion, le serveur vous enverra une première image au format PNG, de 10 pixels de large par 5 pixels de haut. Vous devez renvoyer, en moins de 1000 ms, le code RGB du pixel moyen (moyenne faite couleur par couleur), encodé sur trois octets. Le serveur vous enverra alors un message indiquant que vous avez réussi (« SUCCESS »), que vous avez échoué (« FAIL »), ou que vous avez mis trop de temps à répondre (« TIMELIMIT »). Vous aurez ainsi à répondre à 50 questions de ce type. Si vous répondez 30 fois dans les temps et avec succès, le serveur vous enverra un paquet contenant un lien vers un code secret. À l'issue de ce challenge, ou si vous mettez beaucoup trop longtemps à répondre, le serveur vous enverra un paquet (« GOODBYE »).
Pour ne pas trop compliquer, vous pouvez faire les suppositions suivantes sur les images que le serveur envoie : RGB, pas de transparence, pas d'entrelacement, et un « bit depth » de 8.

1. Codez un programme qui envoie un paquet `Hello` au serveur, et reçoit la réponse appropriée [6], signifiant que le serveur est en ligne.
2. Améliorer votre programme pour qu'il ferme la connexion et termine quand il a reçu un paquet « GOODBYE ».
3. Codez une fonction qui, étant donné une image, renvoie le code de la couleur majoritaire.
4. Finissez votre programme de sorte qu'il obtienne le code secret. Quel est ce code ?

Exercice 3. *Matrix*

Cet exercice a pour but de simuler la transmission de données par un canal radio.
L'ensemble du code produit devra l'être dans le langage C (mis à part MAKEFILES et autres scripts de visualisation ou d'automatisation de tests qui pourront être écrit en BASH [7] ou PYTHON3).

Pour transmettre un signal (par exemple composé de 0 et de 1) sous la forme d'une onde radio, il est nécessaire de le transformer. En pratique, on utilise une onde appelée onde porteuse dont on fait varier les paramètres tels que la fréquence, la phase, ou l'amplitude, afin de transporter des informations. Ce procédé s'appelle la modulation.
Ainsi, on peut représenter une onde radio s sous la forme suivante, à l'instant t : $s(t) = A(t)\cos(2\pi f(t)t + \phi(t))$.
$A(t)$ est l'amplitude de l'onde, $f(t)$ est la fréquence de cette onde, et $\phi(t)$ sa phase à l'origine.
Chaque grandeur est donc susceptible d'évoluer en fonction du temps, selon la modulation choisie.

Pour des raisons qui ne sont pas détaillées ici, au lieu de représenter un signal à l'instant t par la valeur $s(t)$, on utilise plutôt une paire de flottants (I, Q) représentant un nombre complexe $I + i * Q$, dont l'argument représente la phase instantannée du signal et le module l'amplitude.

1. Écrivez un programme qui prend en entrée (à l'aide d'un pipe du shell) un message utilisant des caractères couverts par la norme ASCII de taille arbitraire et renvoie une chaine de caractères composée de '0' et de '1' représentant ce message.
   Par exemple, lancé dans un shell BASH,

   ```
   echo "Bonjour" | ./programme_1
   ```

   devra renvoyer la chaine de caractère "1000010110111111011101101010110111111101011110010000010 10"
2. Lors de la transmission d'un message sous la forme d'un signal radio, il y a parfois des erreurs. Il est donc intéressant d'avoir une certaine forme de redondance dans les messages transmis afin de pouvoir détecter ces erreurs, et éventuellement les corriger. Pour cela, on utilise des codes détecteurs et correcteurs d'erreurs.
   Écrivez un programme qui prend en entrée une chaine de caractères composée de '0' et de '1' de taille arbitraire et qui implémente le code correcteur Hamming(8,4) (aussi appelé Hamming(7,4) avec bit de parité), qui est un type de code correcteur d'erreur.
   Par exemple, lancé dans un shell BASH,

6. World !
7. On accepte aussi les dérivés comme FISH ou ZSH.

```
echo "01001100" | ./programme_2
```

devra renvoyer la chaine de caractère "1001100101111000".

Comment gérer les chaines de caractères dont la taille n'est pas un multiple de 4 ?

3. La modulation de type BPSK, pour Binary Phase Shift Keying, utilise la phase d'une onde radio pour transmettre des 0 et des 1.

   Pour transmettre un bit $n = 0$ ou $n = 1$, on émet donc un signal du type $s_n(t) = A * \cos(2\pi f t + \pi(1 - n))$, qui possède donc deux phases différentes (0 ou π) selon que l'on transmet un 0 ou un 1. On fixe $A = 1$ dans la suite de l'énoncé, et $f = 10$.

   Lors de la transmission d'un signal composé uniquement de 0, à l'aide de la modulation BPSK telle que définie ci-dessus, quel nombre complexe représente l'onde émise ? Et dans le cas d'un signal composé uniquement de 1 ?

4. Écrivez un programme qui prend en entrée une chaine de caractères composée de '0' et de '1' de taille arbitraire et qui implémente le codage BPSK décrit ci-dessus. La sortie du programme sera composée d'une suite de lignes représentant des paires (I, Q) : la première ligne représentera le I correspondant au premier bit du signal, la deuxième ligne le Q correspondant au premier bit du signal, la troisième ligne le I du second bit du signal, etc.

   Les I et Q seront affichés comme des flottants.
   Par exemple, lancé dans un shell Bash,

   ```
   echo "101" | ./programme_3
   ```

   renverra :

   ```
   1.0
   0.0
   -1.0
   0.0
   1.0
   0.0
   ```

5. Écrire un programme qui prend en entrée la sortie du programme qui précède, et ajoute du bruit aléatoire (de moyenne 0 et à valeur dans $[-R; R]$ sur chaque I et chaque Q, où R est un argument représentant un flottant positif pris sur la ligne de commande.
   Par exemple, lancé dans un shell Bash,

   ```
   echo "1.0
   0.0" | ./programme_4 0.0
   ```

   renverra :

   ```
   1.0
   0.0
   ```

   et

   ```
   echo "1.0
   0.0" | ./programme_4 1.0
   ```

   pourra renvoyer :

   ```
   1.4
   -0.8
   ```

6. Écrire l'ensemble des programmes permettant d'utiliser la sortie du programme qui précède pour récupérer le message initial (programme_5 tentera d'enlever le bruit, programme_6 passera du signal modulé par BPSK à une suite de caractères '0' ou '1', et enfin programme_7 détectera les erreurs et les corrigera à l'aide du code correcteur d'erreur utilisé dans programme_2, et enfin programme_8 passera de la représentation binaire à la représentation ASCII.

   Ainsi, exécuté dans un shell comme Bash,

   ```
   echo "Bonjour" | ./programme_1 | ./programme_2 | ./programme_3 | ./programme_4 0.0
   | ./programme_5 | ./programme_6 | ./programme_7 | ./programme_8
   ```

   devra afficher la chaine de caractère "Bonjour".

7. En faisant varier le paramètre du programme_3 entre 0.0 et 5.0, étudier le taux de décodage correct (i.e. on a en sortie ce que l'on a donné en entrée) de la chaine ci-dessus (enchainement de programme_1, programme_2, etc) de messages de tailles différentes (par exemple, 5 caractères, 10 caractères, 50 caractères, 250 caractères, et 500 caractères). Représenter les données sur un graphe.

Exercice 4. *NSA*

L'ESP8266 est un circuit intégré à microcontrôleur avec connexion WiFi développé par le fabricant chinois Espressif. [8].

Il est possible de programmer ce microcontrôleur avec le langage C/C++ à l'aide de l'IDE arduino (`https://www.arduino.cc/en/main/software`), par exemple en suivant les instructions du paragraphe "Configuration de l'IDE Arduino" disponibles à l'adresse `https://www.fais-le-toi-meme.fr/fr/electronique/tutoriel/programmes-arduino-executes-sur-esp8266-arduino-ide`.

Des exemples de code pour l'esp8266 sont disponibles à l'adresse `https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi/examples`.

1. Installez l'IDE arduino et configurez le pour parler avec l'esp8266 que l'on vous a fourni.

   Le modèle à utiliser dans le menu "Outils" est le modèle "NodeMCU 1.0 (ESP-12E Module)" (qui est une carte qui intègre l'esp8266 avec un port microUSB pour faciliter sa programmation).

   Ouvrez l'exemple WifiScan (Fichier > Exemples > ESP8266Wifi > WifiScan), compilez le, téléchargez le sur le microcontrôleur et observez son comportement.

   À quoi sert la fonction setup() ? La fonction loop() ? À quoi sert l'instruction "Serial.begin(115200);" ?

2. Modifier l'exemple WifiScan pour qu'il affiche en plus l'adresse MAC des bornes Wifi à portée, et les différents types de chiffrement utilisés.

3. Comment faire pour communiquer des informations entre le NodeMCU et l'ordinateur, sans utiliser des ondes Wifi ? Créez un script python3 qui communique avec le NodeMCU directement, sans passer par l'IDE arduino, et enregistre les données ainsi transférées dans une base de données sqlite3 (la date et l'heure de chaque mesure doit être ajoutée dans la base de donnée).

4. À l'aide de ces données, écrire un script python3 utilisant une API en ligne telle que `https://developers.google.com/maps/documentation/geolocation/intro?hl=fr` pour déterminer la position géographique (GPS) associée à un ensemble de mesures.

8. Source : `https://fr.wikipedia.org/wiki/ESP8266`