Compte-rendu Sécurité :

Réponses aux questions du TP :

Question 1 :

Dans le cadre général d'un virus, préciser quelles sont les grandes fonctions que met en oeuvre un virus lors d'une attaque (mécanisme d'attaque) et leurs enchaînements, fonctions qu'il faudra donc implémenter.

Réponse 1:

1. Un virus doit détecter les fichiers à contaminer
2. S'assurer qu'il ne sont pas déjà contaminé par une version de lui-même
3. Contaminer la cible

Question 2 :

Détailler, dans le cas du virus compagnon (projet), pour chacune de ces fonctions, les étapes qui devront être implémentées.

Réponse 2 :

Le principe du virus compagnon est de se faire passer pour quelque chose. Celui-ci vole l'identité et l'emplacement d'un fichier / cible existant en prenant son nom.
Dans un premier temps, l'objectif de ce virus est de détecter les fichiers à infecter, en lisant les fichiers dans le répertoire courant dans notre cas. Pour s'assurer que le fichier n'est pas déjà contaminé celui-ci doit vérifier si il existe une version du fichier cible avec l'extension "*.old"* ce qui signifie qu'il est déjà infecté et qu'il est alors inutile de poursuivre le traitement d'infection. Dans le cas contraire l'infection doit avoir lieu, le virus devra donc renommer le fichier cible en ajoutant "*.old"* à la fin de son nom d'origine. Ensuite le logiciel malveillant se duplique en lieu et place du fichier infecté et enfin se renomme avec le nom d'origine du fichier cible. Ainsi l'utilisateur pensera exécuté une fichier sain, mais en réalité ce sera dans un premier temps le virus qui sera exécuté en s'assurant de sa propagation. Les soupçons de l'utilisateur seront évité car le virus lancera l'exécution du programme originale par la suite.

Question 3 :

Rendre Service : Écrire en langage C la fonction du virus MediaPlayer.exe permettant d'afficher une image.

Réponse 3 :

Pour cela nous avons utilisé le GdkPixBuff et GtkImage. Nous avons dans un premier temps lu les chemins d'images contenu dans le répertoire *"images"* que nous avons stocké dans un tableau.
Nous récupérons le chemin voulu dans un GdkPixBuff que nous donnons par la suite au GtkImage. Lors d'un clique sur les boutons précédent ou suivant, nous diminuons ou augmentons respectivement la valeur de l'indice à lire dans le tableau avec un modulo pour lire une la nouvelle image.

Question 4 :

Recherche des fichiers cibles (1 fichier cible a le statut d'être exécutable par l'utilisateur et régulier)
Ecrire en langage C une fonction permettant :

1 - d'ouvrir le répertoire courant
2 - accéder aux fichier qui s'y trouvent
3 - pour chaque fichier récupération de leur statut
4 - Afficher le nom des fichiers qui correspondent à la cible

Vérifier le bon fonctionnement du programme constitué de l'union des fonctions développées dans 3. et 4.

Réponse 4 :

La fonction *step1listProg* nous permets de lire et stocker dans une variable de type tableau *executables* le nom des fichiers dans présent de le répertoire courant. Nous avons utilisé la fonction *readdir* pour lire les fichiers du répertoire courant, ils sont conservé si ils ont le statut executable *S_IXUSR* et régulier *S_IFREG*. Nous pouvons donc faire défiler les images (3) et récupérer les fichiers du répertoire courant (4).
Pour chaque éléments du tableau, donc pour chaque fichier cible nous allons lancer l'infection si celui-ci n'est pas infecté. Voir point suivant.

Question 5 :

Lutte contre la surinfection (éviter d'être détecté)
Le fichier courant est-il déjà infecté ?

- Si oui passer à un autre fichier cible
- Sinon infecter ce fichier.

Ecrire en langage C une fonction permettant de tester si le fichier cible courant est déjà infecté.

- Pourquoi d'après vous, dans le cas du virus compagnon cette vérification doit-être double ?

Réponse 5 :

Ecrire en langage C une fonction permettant de tester si le fichier cible courant est déjà infecté.
La fonction *step2VerificationInfection* nous permet de mettre en place la vérification retournant un booléen pour savoir si le fichier est déjà infecté ou non. Pour cela nous avons utilisé la fonction *strstr* permettant de savoir si l'extension "*.old"* est présente dans le nom du fichier. Si c'est la cas le fichier est considéré infecté et l'étape d'infection ne sera donc pas nécessaire.
Dans le cas ou il n'y a pas l'extension *".old"* dans le nom du fichier cible en question il est alors nécessaire de savoir si un autre fichier du même nom existe dans le répertoire avec cette extension, ce qui signifierait que le fichier est déjà infecté. Nous utilisons *fopen* sur le nom du fichier cible concaténé avec *".old"*, si l'ouverture peut se faire c'est que le fichier existe et qu'il est déjà infecté, sinon c'est un fichier sain et il faut le contaminer.

Pourquoi d'après vous, dans le cas du virus compagnon cette vérification doit-être double ?
Si le fichier cible est déjà infecté, il n'y a évidemment pas besoin de le ré-infecter, une vérification doit alors être faite.

1. Si le nom du fichier cible courant possède l'extension "*.old"*, alors le fichier est infecté, on ne fait rien.
2. Si le nom du fichier ne possède pas l'extension "*.old"* alors il faut chercher dans le répertoire si il existe une version du fichier avec l'extension "*.old"*, ce qui signifierait que le fichier est infecté.
    - Si le fichier n'est pas infecté : il faut l'infecter.

La vérification est alors double, vérification de la cible actuelle (1) et vérification dans le dossier courant (2).

Question 6 :

Infection des fichiers cibles non infectés.
L'infection comporte 3 étapes :

- Renommer le fichier cible courant avec l'extension *".old"*
- Dupliquer le virus, pg en cours d'exécution, copie portant le bon nom
- Cette copie doit être en version exécutable

- D'après vous que ce produit-il si l'étape 3 est oubliée ou ne fonctionne pas ?
- D'après vous, comment à travers cette étape (6) peut-on amplifier l'infection ?
- Ecrire en langage C une fonction permettant de réaliser l'infection des fichiers cibles.
- Réalisez l'union des fonctions développées en 3 / 4 / 5 / 6
- Testez son bon fonctionnement en vérifiant dans le répertoire courant la présence de fichiers exécutables doubles (même nom mais avec extension différentes, tous deux exécutables)

Réponse 6 :

Ecrire en langage C une fonction permettant de réaliser l'infection des fichiers cibles.
La fonction *step3infection* nous permet d'infecter le fichier cible. Pour cela on concatène l'extension "*.old"* au nom du fichier cible originale et on utilise la fonction *rename* pour renommer le fichier avec l'extension "*.old"*. Par la suite on duplique le virus en cours d'exécution par le nom original du fichier cible en utilisant la commande *cp* à travers *execl* lancé par un programme parallèle grâce à l'utilisation du *fork*. Ainsi notre virus est alors dupliqué en volant l'identité du fichier cible et tout en restant un fichier exécutable.

D'après vous que ce produit-il si l'étape 3 est oubliée ou ne fonctionne pas ?
Si la copie du virus n'est pas en version exécutable, alors celui-ci ne sera pas capable de s'exécuter et donc de se propager.

D'après vous, comment à travers cette étape (6) peut-on amplifier l'infection ?
Pour amplifier l'infection à travers l'étape 6, on pourrais lorsque la duplication vient de se faire exécuter le nouveau fichier contaminé. Celui-ci peut alors contaminé plus rapidement le répertoire courant. Et pourquoi pas aller infecter d'autres répertoires. Bien sûr ici nous voulons limiter la diffusion du virus mais une recherche depuis le dossier racine pourrait être bien plus destructeur et apporter une vrai charge finale.

Réalisez l'union des fonctions développées en 3 / 4 / 5 / 6
Cette union correspond donc au traitement complet du virus :

3 - Mise en place du logiciel souhaité par l'utilisateur, ici un lecteur d'image, pour éviter les soupçons de celui-ci.
4 - Recherche des fichiers à infecter
5 - Vérification pour éviter la sur-infection et éviter un possible repérage des antivirus.
6 - Infection

Testez son bon fonctionnement en vérifiant dans le répertoire courant la présence de fichiers exécutables doubles
Une fois le virus lancé, tous les programmes du répertoire courant non infecté seront contaminé et posséderont 2 versions exécutables :

- 1 version saine renommer avec l'extension "*.old"*, cet exécutable sera appelé par le virus pour faire croire au bon fonctionnement du programme à l'utilisateur.
- 1 version contaminé qui correspond à la copie du virus et qui sera exécuté par l'utilisateur pensant lancé un programme sain.

Question 7 :

Transfert d'exécution au code hôte :
2 cas peuvent se présenter :

- Si le programme demandé par l'utilisateur est MediaPlayer.exe, après infection des fichiers cibles, il n'est pas nécessaire de transférer le contrôle à un autre programme (exécuter un autre programme). On est dans le cas première infection.

- Par contre, si le programme demandé par l'utilisateur est autre que MediaPlayer.exe, alors il s'agit d'un hôte infecté. Il est alors nécessaire de transférer l'exécution au programme demandé par l'utilisateur.

D'après-vous que ce produit-il si l'étape 7 est oubliée ou ne fonctionne pas ?

Réponse 7 :

Lorsque le programme lancé est MediaPlayer.exe, l'utilisateur s'attend à voir un logiciel de visionneuse d'image. Ce service doit être rendu en plus de l'action malveillante car on ne doit surtout pas éveiller les soupçons dès la première utilisation.
Dans le cas ou le programme lancé n'est pas mediaPlayer.exe, pour que l'utilisateur n'ai pas de soupçons il faut que cette copie du virus lance le programme demandé par l'utilisateur après avoir lancé les étapes d'infections du virus. Le virus va alors en fin de traitement, lancé avec la commande *execl* le nom du programme donné en paramètre en ajoutant l'extension "*.old"* qui est la version original et saine du programme demandé.
Si cette étape est oubliée ou ne fonctionne pas l'utilisateur va se rendre compte d'un problème et peut-être détecter la présence du virus.

Détails de conception du virus et résultats obtenu :

Pour réaliser notre virus nous avons utilisé 4 fonctions principale :

1. Affichage des images
2. Une fonction de récupération des nom de fichier présent dans le répertoire courant : cibles
3. Une fonction de vérification d'infection
4. Une fonction d'infection

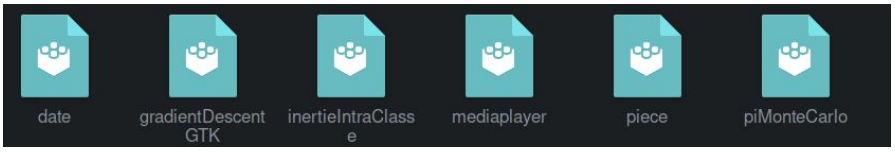
Le virus mediaplayer commence par charger les images dans un tableau et en affiche une. Il est possible de changer d'image de façon à lire toutes les images présentes dans le répertoire d'image, lors du changement, un indice modulo la taille du tableau évolue de façon à afficher l'image voulu.

Le virus lance par la suite la première fonction malveillante (2.) de façon à récupérer les fichiers à infecter dans un tableau si ceux-ci sont exécutables. Les fonctions 3 et 4 s'exécutent pour chacune des cibles.
La fonction 3 vérifie si le fichier est déjà infecté pour éviter de le recontaminer afin d'éviter la sur-infection qui est un potentiel risque de démasquage. Si le fichier n'est pas infecter l'action 4 s'effectue.
La fonction 4 permet d'infecter la cible en volant son identité. Il la renomme en ajoutant l'extension "*.old"*, se duplique et vole le nom original de la cible afin de se faire passer pour le programme cible sain. Mediaplayer n'est pas affecté, celui-ci étant le virus originale.

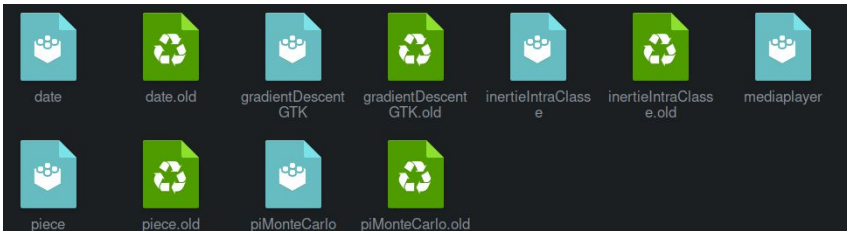
Pour finir si le programme lancé est MediaPlayer.exe alors c'est la primo-infection et le programme se termine. Dans le cas contraire c'est le virus qui se fait passer pour un autre programme et il faut aller exécuter le programme que l'utilisateur pense exécuter de façon à ne pas éveiller les soupçons. Pour cela nous utilisons les commandes *fork* et *execl* qui créer un autre processus qui exécutera le programme original portant l'extension "*.old"*.

Résultats :

Fichier Sain, virus non exécuté



Virus lancé, infection des fichiers cibles



Ajout d'un nouveau programme et lancement d'un utilitaire infecté, infection du nouveau programme par la copie du virus.

Fonctionnement des utilitaires :

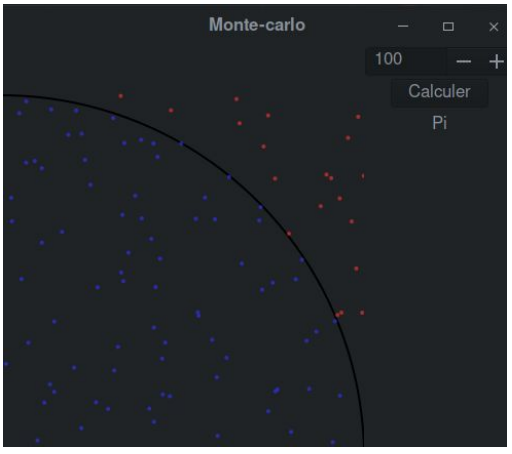
Nous avons mis en place 5 utilitaires :

1. Calcul de pi avec la méthode de Monte Carlo : Version Gtk
2. Calcul de la descente de gradient pour la régression linéaire : Version Gtk
3. Calcul de l'inertie intra-classe entre les points d'un groupe.
4. Calcul de différence de date
5. Jeu de hasard pile ou face.

Lancement des utilitaires et de mediaplayer: *make* *cd build* *./nom*

1. Calcule de Pi avec la méthode de Monte Carlo :

Nous avons mis en place le calcule de pi en utilisant la méthode de monte-carlo. C'est une façon de calculer pi basé sur le hasard. On "lance" au hasard des points dans la zone de dessin et on compte le nombre de point dans un cercle de rayon 1 en partant de x = 0 et y = 0, le tout à l'échelle de la fenêtre. Avec un calcule simple utilisant pythagore on sais si le point est dans le cercle ou non. Pour finir on calcule pi = (4*nombre de point de le cercle) / le nombre de points total.
Nous avons utilisé gtk pour afficher en bleu les points dans le cercle et en rouge les points en dehors du cercle. Il est possible de choisir le nombre de points à "lancer".



2. Calcule de la descente de gradient pour la régression linéaire :

La Descente de Gradient est un algorithme d'optimisation qui permet de trouver le minimum de n'importe quelle fonction convexe en convergeant progressivement vers celui-ci. Il faut alors minimiser la fonction coût. Dans le cas de la régression linéaire la fonction coût à minimiser est l'erreur quadratique moyenne.

$$J(a,b) = \frac{1}{2m} \sum_{i=1}^{m} (ax^{(i)} + b - y^{(i)})^2.$$

Nous avons plusieurs points placés au hasard dans la zone de dessin. Le but est de trouver une droite permettant de passer au plus proches de ses points.
Les valeurs a et b sont les 2 paramètres de la droite y = ax+b qui seront calculé avec cet algorithme. Plus la fonction coût sera faible meilleur sera le modèle et plus la droite sera proche des points. Il faut donc minimiser cette fonction coût.

On calcule la dérivé partielle par rapport à a :

$$\frac{\partial J(a,b)}{\partial a} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \times (ax^{(i)} + b - y^{(i)})$$

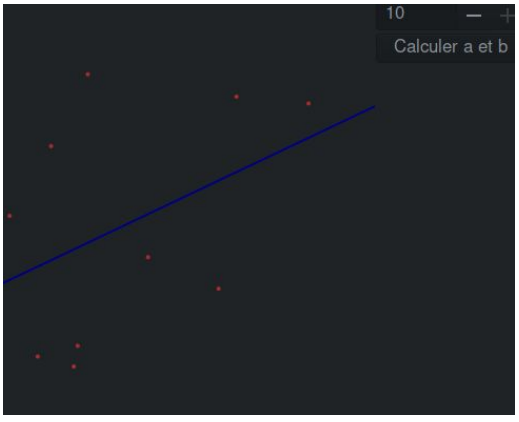
et la dérivé partielle par rapport à b :

$$\frac{\partial J(a,b)}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (ax^{(i)} + b - y^{(i)})$$

On choisi un learning rate α faible pour être sur d'arriver au plus proche du minimum.

$$\begin{cases} a := a - \alpha \dfrac{\partial J(a,b)}{\partial a} \\ b := b - \alpha \dfrac{\partial J(a,b)}{\partial b} \end{cases}$$

On calcule a et b tant que qu'il n'y a pas convergence.
Nous arrivons au final avec 2 paramètres a et b permettant d'obtenir une droite qui passe au plus près des points.



3. Calcule de l'inertie-intra classe entre les points d'un groupe :

L'inertie intra-classe est le critère optimisé par l'algorithme C-means, qui a pour but de faire de la classification non supervisé. Plus les points d'un même groupe sont proches les un-des-autres pour l'inertie intra classe est faible.

Exemple 1:
Nombre de groupe : 3
- Nombre de point dans le groupe 1 : 2
  - (5, 2)
  - (3, 7)
- Nombre de point dans le groupe 2 : 1
  - (7, 2)
- Nombre de point dans le groupe 3 : 3
  - (2, 4)
  - (3, 5)
  - (7, 3)

Résultat : 5.17

Exemple 2 :
Nombre de groupe : 3
- Nombre de point dans le groupe 1 : 2
  - (5, 2)
  - (5, 2)
- Nombre de point dans le groupe 2 : 1
  - (7, 2)
- Nombre de point dans le groupe 3 : 3
  - (2, 4)
  - (2, 4)
  - (2, 4)

Résultat : 0.00

Voici la formule utilisé :

$$\mathcal{D}(U,V) = \frac{1}{n}\sum_{i=1}^{c}\sum_{x_k \in C_i} d_2^2(x_k, \overline{x}_i).$$

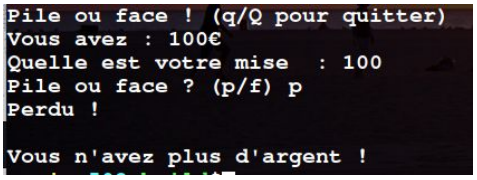
Nous avons donc réalisé plusieurs fonctions pour faire ce calcule tel que :
- Le calcule de distance euclidienne
- Le calcule des centres des groupes.

4. Calcule de la différence de date :

Ce programme permet à l'utilisateur d'entrer une date et l'utilitaire affiche le temps d'écart entre la date actuelle et la date entrée. Nous avons pris en compte les années bisextiles et la durée des mois variable entre 30 et 31 jours.

5. Jeu de pile ou face :

Cet utilitaire correspond à un petit jeu de hasard, l'utilisateur possède 100€ de départ et dois misé sur pile ou face. Si il a juste il gagne sa mise sinon il la perd. Nous utilisons la fonction *rand()%2* pour simuler l'état de la pièce, 0 étant face et 1 pile.



Conclusion :

Le virus fonctionne à 99% du travail demandé, en effet quelque erreurs mémoire sont visibles avec l'outils *valgrind*.
Le programme mediaplayer est donc un virus de type compagnon volant l'identité des fichiers cibles (utilitaires) en faisant croire à l'utilisateur au bon fonctionnement du programme exécuté.