

Réalisé par :

Dumalin Damien

Waterlot Aeton

**Ouanacraille**

**Projet de programmation**



*“A lutter avec les mêmes armes que ton ennemi, tu deviendras comme lui.”*

*Friedrich Nietzsche*

Table des matières

[Introduction : 3](#_Toc123574430)

[Architecture de notre programme : 4](#_Toc123574431)

[Nos différentes fonctions principales : 5](#_Toc123574432)

[Les différentes parties des fonctions : 7](#_Toc123574433)

[Le parcours de fichiers (récursif) : 7](#_Toc123574434)

[L’ouverture d’un fichier : 8](#_Toc123574435)

[La vérification d’ouverture : 8](#_Toc123574436)

[Ajout de l’extension .encr sur le nom des fichiers : 9](#_Toc123574437)

[Boucle de chiffrement : 9](#_Toc123574438)

[Suppression de l’extension, réécriture du nom du fichier : 11](#_Toc123574439)

[Les sockets : 12](#_Toc123574440)

[Séances de laboratoire : 15](#_Toc123574441)

[Conclusion : 16](#_Toc123574442)

[Sources : 17](#_Toc123574443)

# Introduction :

Dans le cadre de notre cours de programmation, nous avons dû réaliser un Ransomware.

Selon Malwarebytes.com, un ransomware est un type de malware qui empêche les utilisateurs d’accéder à leur système ou à leurs fichiers personnels et exige le paiement d’une rançon en échange du rétablissement de l’accès.

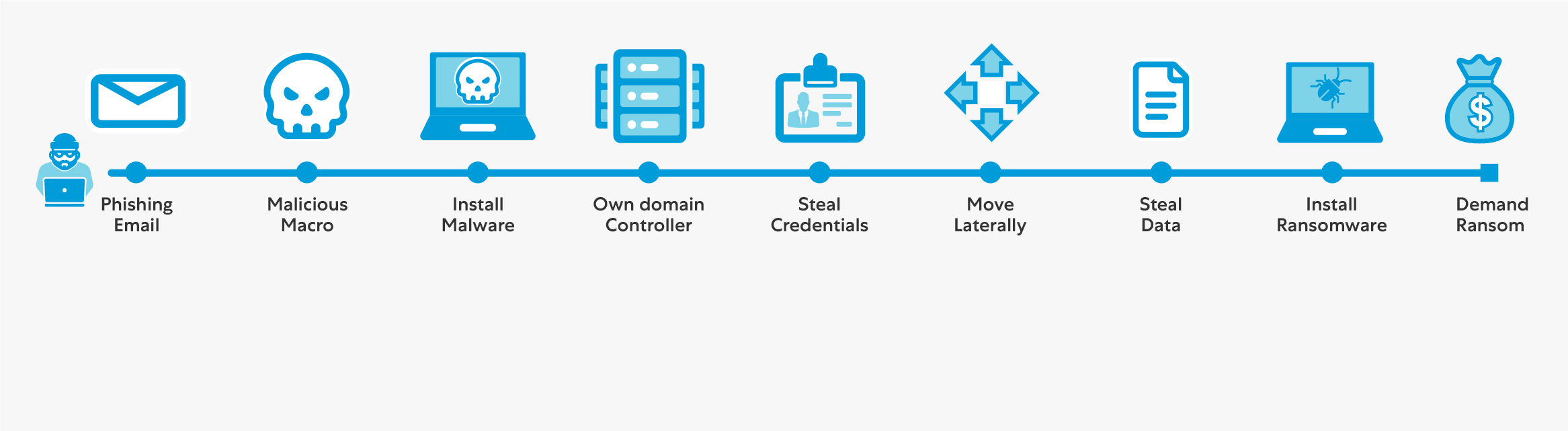
Le ransomware est également illégal dans de nombreux pays, ce qui en fait un acte criminel. En outre, le fait de payer la rançon ne garantit pas toujours que les données seront déchiffrées, ce qui peut entraîner des pertes financières supplémentaires pour les victimes. En raison de ces risques et conséquences négatifs, il est important de se protéger contre les ransomwares et de les éviter à tout prix.

Les premiers ransomwares sont apparus dans les années 1980. A cette époque en l’absence de paiements en ligne, les rançons devaient être envoyées par carte postale. Aujourd’hui, les auteurs de ransomwares, demandent d’être payés en cryptomonnaies ou carte de crédit.

C’est en 2017 que le roi des ransomwares fait son apparition, Wannacry. Il a touché plus de 300 000 ordinateurs à travers le monde, dans plus de 150 pays. La méthode de propagation utilisée était principalement par email avec une pièce jointe piégée. Cette méthode connue s’appelle l’ingénierie sociale.

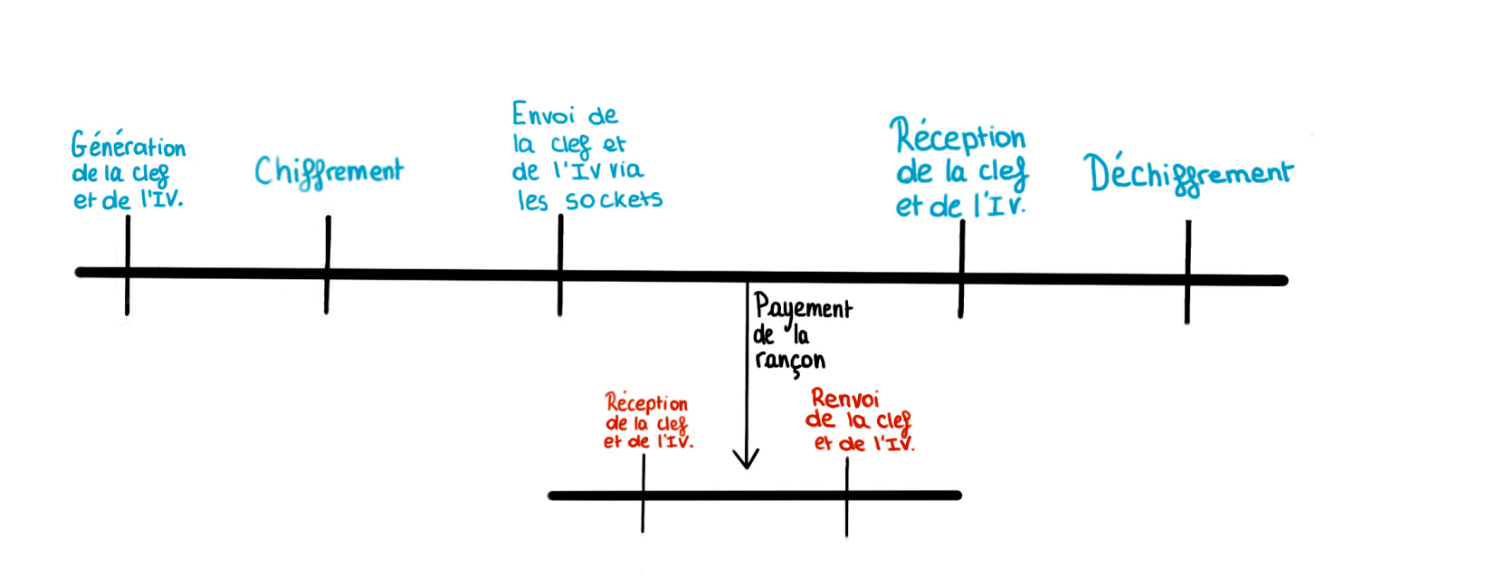
C’est dans ce rapport que nous allons vous présenter notre ransomware créé de toutes pièces à travers les différents laboratoires et grâce à certaines librairies, et fonctions déjà fournies.

Une représentation linéaire des étapes d’un ransomware.

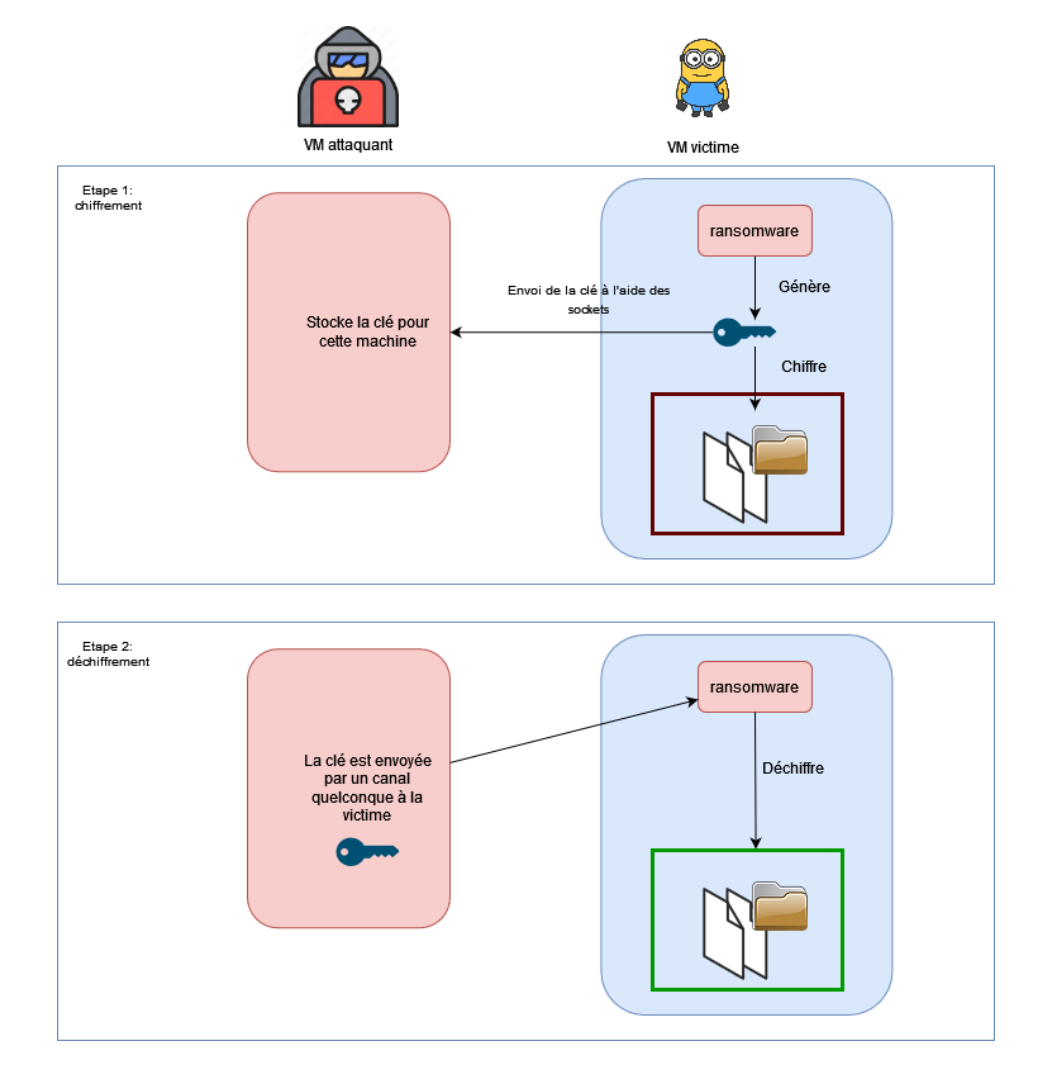


# Architecture de notre programme :

Nous avons imaginé une droite en guise d’architecture de notre programme, qui montre les étapes que notre ransomware suit afin d’arriver à son objectif.



Notre vision se rapproche de l’architecture donnée dans le cours :



# Nos différentes fonctions principales :

int main(int argc, char argv[]);

La fonction main en C est la fonction principale d'un programme. C'est le point de départ de notre programme.

Cette fonction comprend ces éléments suivants :

* int argc : C’est un entier qui représente le nombre d’arguments passés au programme lors de son exécution.
* char argv[] : C’est un tableau de chaine de caractères, qui contient les arguments passés au programme lors de son exécution. Chaque élément du tableau correspond à un argument et argv[] détient le nom du programme en lui-même.

int read\_encrypt(char \*repertory, int depth);

Cette fonction va parcourir tous les fichiers et les dossiers à partir du chemin que l’on a spécifié dans notre code et qui pour chaque fichier va utiliser la fonction encrypt qui chiffrera le fichier à l’aide de notre clef et de notre IV que l’on a généré auparavant.

int read\_decrypt(char \*repertory, int depth);

C’est le même fonctionnement que notre fonction read\_encrypt mais appliquera la fonction decrypt aux fichiers.

int encrypt(unsigned char \*plaintext, int plaintext\_len, unsigned char \*key, unsigned char \*iv, unsigned char \*ciphertext);

Cette fonction provient de OpenSSL. Elle permet de chiffrer un fichier en entrée et de sortir une version chiffrée à l’aide de la clef et de l’IV passés en paramètres.

void handleErrors(void);

Cette fonction permet de faire un « print » affichant l’erreur durant l’exécution de notre code et nous permet d’effectuer un débogage.

int decrypt(unsigned char \*ciphertext, int ciphertext\_len, unsigned char \*key, unsigned char \*iv, unsigned char \*plaintext);

Même fonctionnement que encrypt mais cette fois ci il aura comme entrée notre fichier chiffré et sortira une version déchiffrée à l’aide de la clef et de l’IV passés en paramètres.

int send\_key\_iv(unsigned char \*key, unsigned char \*iv);

Cette fonction permet d’ouvrir un socket sur l’adresse IP de la machine et une fois qu’une connexion est établie à une autre machine alors il enverra le contenu de la variable key et IV à la machine qui s’est connectée.

int receive\_key\_iv(unsigned char \*recup\_key, unsigned char \*recup\_iv);

Celle-ci permet d’effectuer une demande de connexion à une machine et de récupérer ce que l’autre machine nous envoie et de stocker les données reçues dans une variable key et IV.

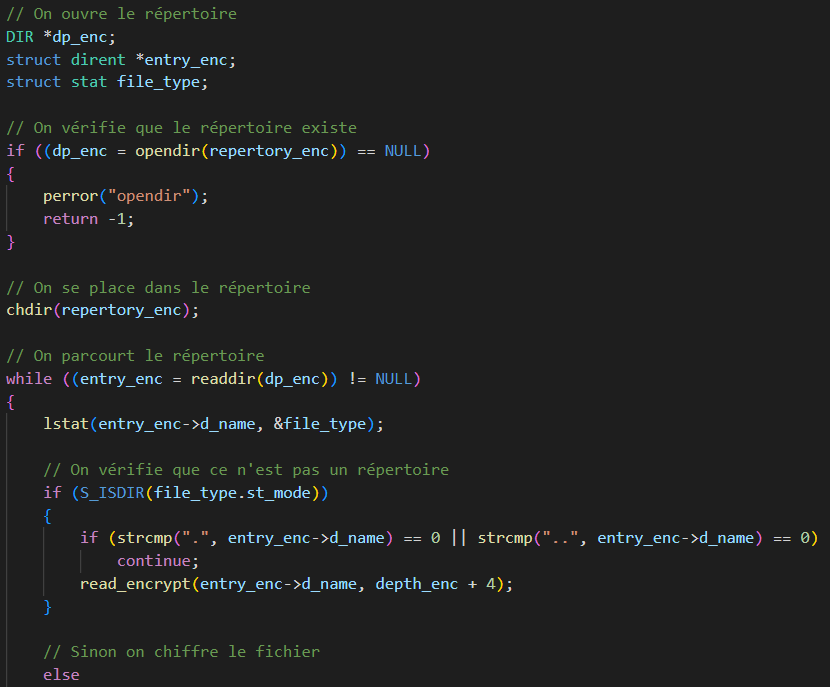
Nous avons besoin des fonctions send\_key\_iv et receive\_key\_iv afin de pouvoir supprimer la clef et l’IV de la machine de la victime pour empêcher le déchiffrement des fichiers mais de les récupérer sur notre machine pour pouvoir les lui rendre une fois le paiement de la rançon effectué et de lancer l’opération de déchiffrement sur sa machine.

Pour les 3 fonctions reprenant, encrypt, decrypt et handleError, elles sont reprises depuis un lien donné dans les PDF du cours.

# Les différentes parties des fonctions :

## Le parcours de fichiers (récursif) :

À l’aide de la libraire <dirent.h>, nous avons su créer un code qui permet de parcourir récursivement un chemin d’accès donné et de traiter les fichiers qui s'y trouvent.



La première ligne déclare un pointeur sur un type DIR, qui est une structure utilisée par les fonctions de gestion de répertoires de la bibliothèque C.

La ligne suivante déclare un pointeur sur une structure dirent, qui est une structure utilisée pour stocker les informations sur un fichier ou un répertoire lors de la lecture d'un répertoire.

La troisième ligne déclare une structure stat, qui est utilisée pour stocker des informations sur un fichier ou un répertoire.

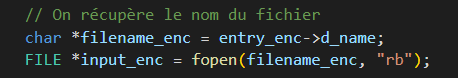
Notre premier if va ouvrir le dossier demandé à l’aide de la fonction opendir() et l’associera a une variable dp\_enc et si l’opération échoue (vérification à l’aide de ==NULL) alors le programme renverra un message d’erreur à l’aide de la fonction perror et renvoie -1.

La fonction chdir() permet de changer le répertoire courant pour celui spécifié en argument.

Ensuite nous ouvrons une boucle while qui utilise la fonction readdir() pour lire les entrées du répertoire, une par une, jusqu'à ce qu'elle renvoie NULL, indiquant la fin du répertoire, dans cette même boucle nous allons faire appel à la fonction lstat() qui va dire si l’entité analysée est un répertoire ou un fichier et va le mettre sur une variable struct.

Ensuite le if S\_ISDIR() va analyser la variable struct de notre entité et si elle correspond à un dossier et n’est pas ‘.’ ou ‘..’ correspondant au répertoire courant et parent alors nous rentrons dans le if et nous appelons récursivement read\_encrypt avec le nom du répertoire ce qui permet de recommencer l’analyse dans un sous-répertoire, sinon il passe au else et nous y appliqueront notre chiffrement par après car tout ce qui peux passer dans le else sera forcément un fichier vu que nous avons filtré les répertoires auparavant.

## L’ouverture d’un fichier :



Ce code ouvre un fichier en mode binaire et en lecture seule.

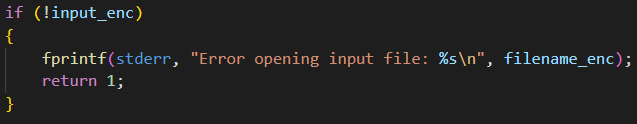
char \*filename\_enc est un pointeur sur un tableau de caractères contenant le nom du fichier à ouvrir.

entry\_enc->d\_name est le nom du fichier pointé par l'objet entry\_enc.

FILE \*input\_enc est un pointeur sur un objet de type FILE, qui représente un flux de données générique pour l'opération in/out.

fopen(filename\_enc, "rb") appelle la fonction fopen(), qui ouvre le fichier spécifié par filename\_enc en mode binaire et en lecture seule ("rb"). Si l'opération réussit, fopen() renvoie un pointeur sur l'objet FILE correspondant, qui est affecté à input\_enc. Si l'opération échoue, fopen() renvoie NULL.

## La vérification d’ouverture :



Cette instruction vérifie si l'ouverture du fichier a échoué.

Si input\_enc est égal à NULL, cela signifie que l'ouverture du fichier a échoué et la fonction affiche un message d'erreur sur la sortie d'erreur standard (stderr).

fprintf() est une fonction qui permet d'afficher du texte formaté sur un flux de sortie, ici la sortie d'erreur standard (stderr).

Le premier argument est le flux de sortie, le deuxième est la chaîne de format (contenant des codes de format %s qui seront remplacés par l’argument suivant) et l’argument suivant est la valeur à afficher à la place des codes de format.

Le message affiché est : "Error opening input file: [nom du fichier]"

Ensuite, la fonction renvoie la valeur 1 avec return 1; indiquant qu'une erreur s'est produite.

## Ajout de l’extension .encr sur le nom des fichiers :



sprintf() est une fonction qui permet de générer une chaîne de caractères formatée à partir d'un format et d'arguments.

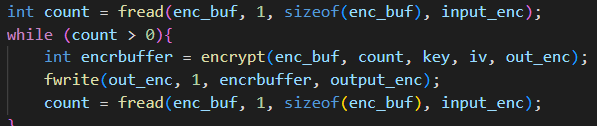
Le premier argument est un pointeur sur une chaîne de caractères où le résultat sera stocké, ici dans la variable output\_filename.

Le deuxième argument est une chaîne de caractère qui correspond à un suffixe.

L’argument suivant est la valeur de la variable introduite + le suffixe entré juste avant.

Dans ce cas précis, sprintf() génère une chaîne de caractères avec le format suivant : [nom du fichier].encr et la stocke dans output\_filename. Par exemple, si filename\_enc contient "image.png", output\_filename contiendra "image.png.encr".

## Boucle de chiffrement :



Cette partie est pour nous la plus importante de notre programme. C’est ici que le contenu des fichiers sera chiffré.

Ce code lit le contenu du fichier ouvert dans input\_enc par blocs de taille sizeof(enc\_buf), où enc\_buf est défini plus haut, dans notre cas ce sera par bloc de 8. C’est après qu’il l'écrit dans un autre fichier ouvert dans output\_enc après l'avoir chiffré à l'aide de la fonction encrypt().

fread() est une fonction qui lit un nombre d'octets d'un fichier et les stocke dans un tampon de mémoire.

Le premier argument est un pointeur sur le tampon de mémoire où les données seront stockées, ici enc\_buf.

Le deuxième argument est la taille en octets d'un élément à lire, ici 1.

Le troisième argument est le nombre d'éléments à lire, ici la taille du tampon enc\_buf.

Le quatrième argument est le pointeur sur l'objet FILE du fichier à lire, ici input\_enc. Si la lecture a réussi, fread() renvoie le nombre d'éléments lus. Si la lecture a échoué ou si la fin du fichier est atteinte, fread() renvoie un nombre inférieur à sizeof(enc\_buf).

fwrite() est une fonction qui écrit un nombre d'octets dans un fichier à partir d'un tampon de mémoire.

Le premier argument est un pointeur sur le tampon de mémoire contenant les données à écrire, ici out\_enc (défini plus haut dans le code).

Le deuxième argument est la taille en octets d'un élément à écrire, ici 1. Le troisième argument est le nombre d'éléments à écrire, ici le nombre d'octets chiffrés renvoyés par la fonction encrypt().

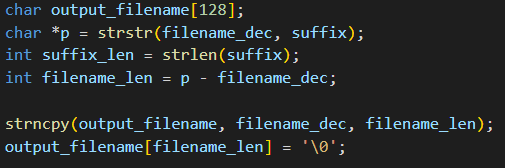
Le quatrième argument est le pointeur sur l'objet FILE du fichier à écrire, ici output\_enc. Si l'écriture a réussi, fwrite() renvoie le nombre d'éléments écrits. Si l'écriture a échoué, fwrite() renvoie un nombre inférieur à celui attendu.

La boucle while s'exécute tant que la valeur de count est supérieure à 0. A chaque itération, le contenu du tampon enc\_buf est chiffré à l'aide de la fonction encrypt(), qui renvoie le nombre d'octets chiffrés et les stocke dans le tampon out\_enc. Le contenu du tampon out\_enc est alors écrit dans le fichier ouvert dans output\_enc à l'aide de fwrite().

On retrouvera la même fonction dans la partie read\_decrypt pour déchiffrer de la même façon.

## Suppression de l’extension, réécriture du nom du fichier :

Dans la fonction où l’on déchiffre, on retrouvera ce bout de code :



Ce code crée un nom de fichier de sortie en enlevant le suffixe .encr qui est stocké dans la variable suffixe, du nom de fichier de départ qui est stocké dans filename\_dec.

strstr() est une fonction qui cherche la première occurrence d'une sous-chaîne dans une chaîne de caractères.

Le premier argument est la chaîne de caractères où la variable dans laquelle chercher, ici filename\_dec.

Le deuxième argument est la partie de chaine de caractères à rechercher. Si la sous-chaîne est trouvée, strstr() renvoie un pointeur sur le début de la sous-chaîne dans la chaîne de caractères, sinon NULL.

strlen() est une fonction qui renvoie la longueur d'une chaîne de caractères (c'est-à-dire le nombre de caractères de la chaîne), excluant le caractère de fin de chaîne '\0'.

strncpy() est une fonction qui copie une partie d'une chaîne de caractères dans une autre chaîne de caractères.

Le premier argument est la chaîne de destination, ici output\_filename.

Le deuxième argument est la chaîne de source, ici filename\_dec.

Le troisième argument est le nombre de caractères à copier, ici filename\_len, qui est la longueur du nom de fichier de départ jusqu'au suffixe.

Ainsi, strncpy() copie les filename\_len premiers caractères de filename\_dec dans output\_filename.

Ensuite, le caractère de fin de chaîne '\0' est ajouté à la fin de output\_filename pour indiquer la fin de la chaîne.

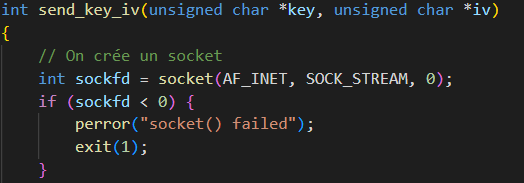
Le résultat final est un nom de fichier de sortie sans le suffixe .encr dans output\_filename.

## Les sockets :

Communication via les sockets pour le transfert de la clef et de l’IV.

Pour réaliser cette partie nous utilisons la librairie arpa/inet.h afin d’effectuer des fonctions sur les sockets.

Tout d’abord nous créons notre socket à l’aide de la fonction "socket ()". Elle prend en entrée 3 arguments :

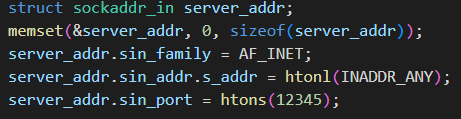


AF\_INET : une constante qui spécifie que le socket sera utilisé pour communiquer avec des hôtes sur Internet

SOCK\_STREAM : une constante qui indique que le socket sera utilisé pour une communication en mode connecté (TCP)

0 : un paramètre optionnel qui peut être utilisé pour spécifier un protocole spécifique à utiliser avec le socket. Dans ce cas, 0 indique que le protocole par défaut (IP) sera utilisé.

Ensuite nous allons faire notre structure sockaddr\_in qui stockera l’adresse ip notre socket et le type du socket en ayant comme variable server\_addr.



On commence par la fonction "memset()" qui remplit la mémoire allouée pour la structure "sockaddr\_in" avec des zéros, afin de s'assurer que tous les champs de la structure sont initialisés correctement. Elle prend en entrée 3 arguments :

&server\_addr : l'adresse de la structure "sockaddr\_in" à remplir avec des zéros.

0 : le caractère à utiliser pour remplir la mémoire (ici, 0).

sizeof(server\_addr) : la taille de la mémoire à remplir en octets.

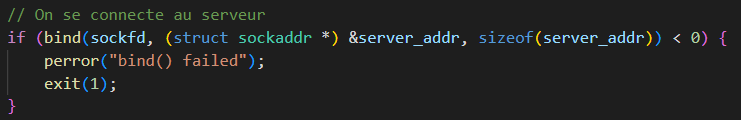
Après avoir fait cela nous pouvons commencer à donner nos différentes données pour la structure :

‘server\_addr.sin\_family = AF\_INET;’ correspond au type du port et AF\_INET correspond à un type de port pouvant aller sur internet.

‘server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);’ ceci correspond à l’adresse de notre serveur qui serra l’adresse de notre machine.

‘server\_addr.sin\_port = htons(12345);’ pour finir nous définissons le numéro du port que nous allons utiliser.

La fonction "bind()" nous permet d’associer le socket à notre adresse IP et notre port. Elle prend en entrée 3 arguments :

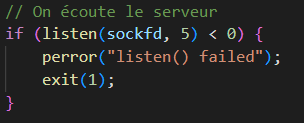


sockfd : le descripteur de socket retourné par la fonction "socket()"

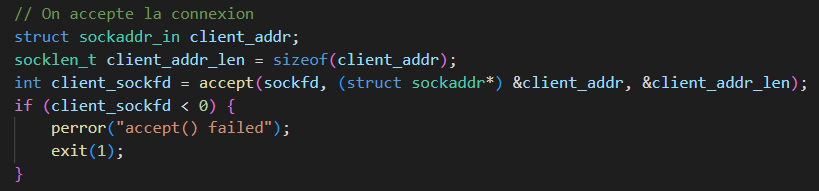
server\_addr :notre structure "sockaddr\_in"

sizeof(server\_addr) : la taille de la structure "sockaddr\_in" en octets

Ensuite nous lançons l’écoute sur le port à l’aide de la fonction listen() qui prend en argument notre port(sockfd) ainsi que le nombre maximal de connexions entrantes dans la file d’attente.

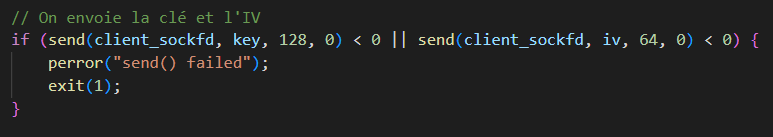


Pour accepter les connexions au port nous utilisons la fonction accept()



Nous faisons donc pareil que pour notre struct sockaddr\_in à l’exception que nous mettrons les données sur notre variable à l’aide de la fonction accept qui prendra en argument le port utilisé ainsi qu’un pointeur sur la variable struct du client et ensuite l’adresse mémoire de l’IP et la longueur de l’IP.

Ensuite nous envoyons les données via send() qui prend comme variable le struct du client créé à l’aide du accept(),le pointeur sur le tableau d'octets contenant les données envoyées, la taille du tableau en octet et un paramètre optionnel (flag).

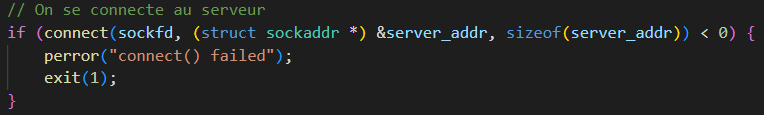


Réception de fichier via socket.

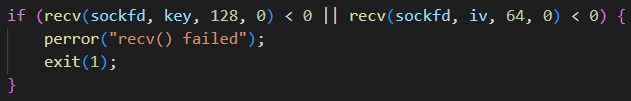
Nous commençons par créer un socket de la même manière que pour l’envoi.

Nous définissons la même struct sockaddr\_in que pour l’envoi à l’exception de l’\_addr qui ne correspondra plus à INADDR\_ANY mais qui serra l’adresse IP de la machine qui doit nous envoyer la clef et l’IV.

Ensuite à la place du accept nous faisons un connect() contenant les même arguments que l’accept à l’exception du nom donné aux variables qui seront « serveur » à la place de « client ».



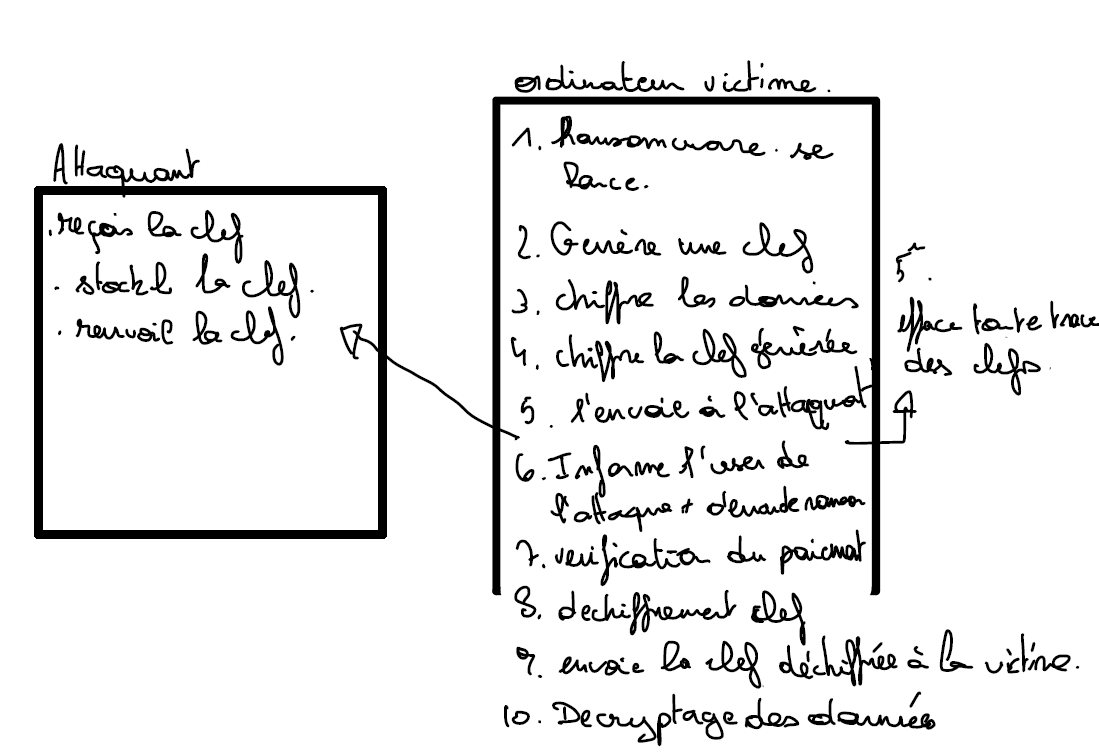
Pour finir nous effectuons un recv() qui permet de recevoir les données et qui auront exactement les mêmes arrangements d’arguments que send(), effectué lors de l’envoi.



# Séances de laboratoire :

Chaque étape des laboratoires nous faisait évoluer dans la connaissance du langage C ainsi que sa compréhension.

Pour la première séance de laboratoire, nous avons dû réaliser un schéma qui reprendrait l’ordre, la marche à suivre qu’on pensait être la bonne. Voici le schéma réalisé :



Ensuite, nous avons travaillé, appris la base de notre projet, la récursivité. Pour ensuite pouvoir apprendre l’utilité des pointeurs, qui permettent de travailler avec des valeurs inscrites en mémoire sur base de leur adresse mémoire.

Après, nous sommes arrivés sur une autre partie très importante qui comprend le chiffrement des données grâce à des fonctions extraites de la librairie d’OpenSSL.

Enfin nous avons terminé sur les sockets, qui nous permettent dans notre cas d’envoyer la clef de chiffrement ainsi que l’IV via le réseau.

C’est en plus, avec l’aide d’internet et les cours théoriques que nous sommes arrivés à la fin de notre ransomware.

# Conclusion :

En conclusion, nous avons réalisé un ransomware, qui est une sorte de malware qui bloque l’accès à des données jusqu’à la réception d’une rançon.

C’est lors des séances de laboratoires que la plupart de nos réflexions se sont faites à l’aide des outils fournis tout au long des étapes. Chaque étape nous a permis de comprendre point à point différentes parties de notre code.

Pour notre programme nous avons pensé à 3 améliorations. La première une des plus importantes, est de simplifier en une fonction le parcours récursif des fichiers afin de ne pas se répéter 2 fois la même chose dans notre code.

La deuxième serait de lancer automatiquement le déchiffrement une fois que la partie attaquante confirme la réception de la rançon.

La dernière consiste à chiffrer la clef en RSA avant l’envoi de celle-ci via les sockets.

# Sources :

Voici quelques sources que nous avons pris la peine de noter.

<https://wiki.openssl.org/index.php/EVP_Symmetric_Encryption_and_Decryption>

<https://www.w3resource.com/c-programming-exercises/file-handling/c-file-handling-exercise-14.php>

<https://www.youtube.com/watch?v=7d7_G81uews&t=116s>

<https://public.iutenligne.net/informatique/algorithme-et-programmation/priou/LangageC/72_le_mode_demploi_dune_fonction__le_prototype_ou_dclaration.html>

<https://chat.openai.com/>

<https://adrien.poupa.net/efrei/Programmation%20en%20Java/Redaction_Rapport.pdf>

<https://fr.wikipedia.org/>