Ransomware Documentation

> *fait par Thomas et Simon*

Introduction

Ce document contient la documentation ainsi que la façon d'utiliser le ransomware. Veuillez noter qu'un ransomware est un programme malveillant et qu'il est donc fortement déconseillé de l'utiliser sans en connaitre son fonctionnement.

Celui-ci est un programme qui a été créé dans un but éducatif.
Nous révoquons toutes responsabilités quant à l'usage que vous pourriez en faire.

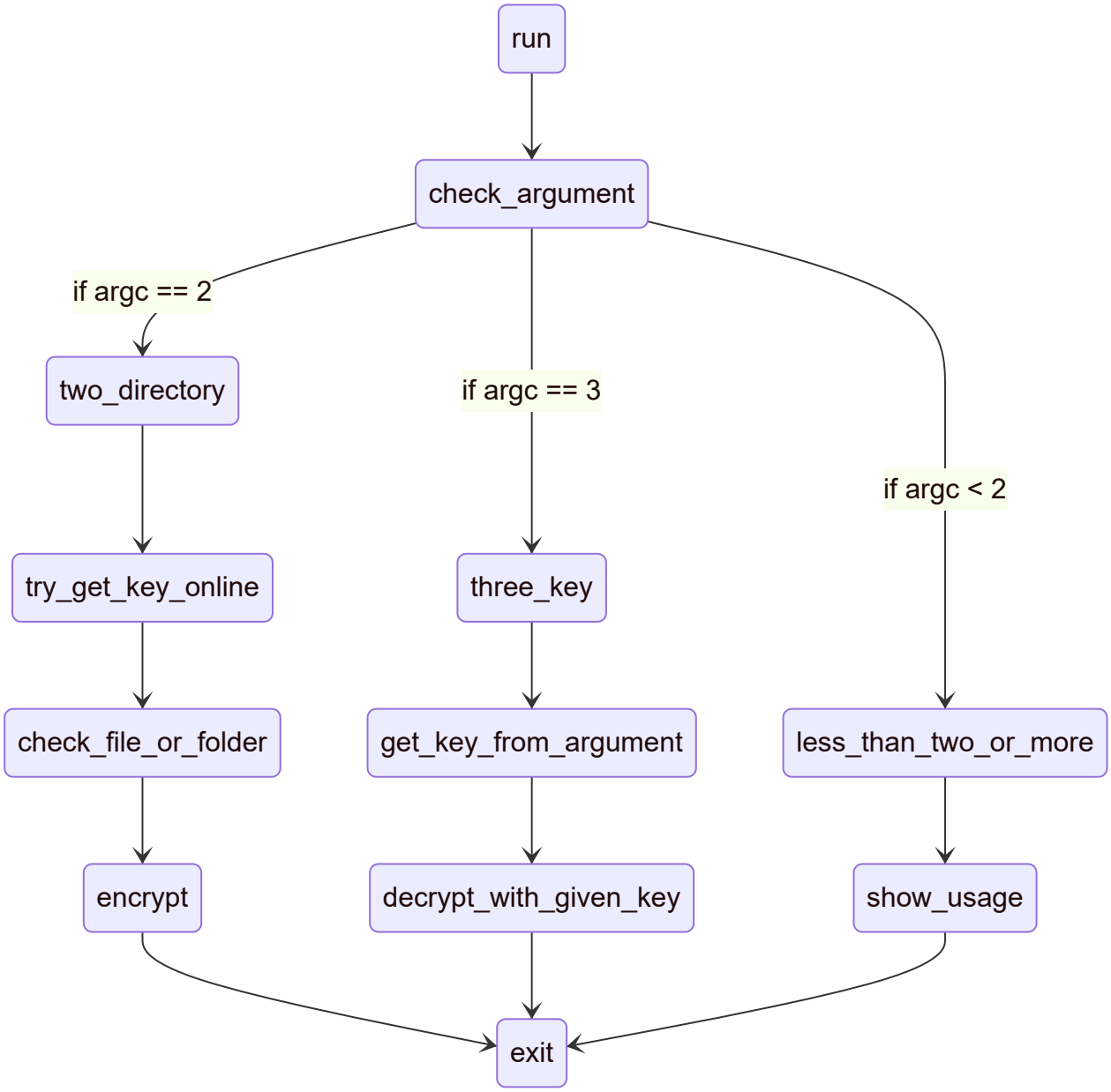
Qu'est-ce qu'un ransomware

Un ransomware est un programme malveillant ayant pour but de couper l'accès à certaines données en les chiffrant afin que la personne victime du programme ne puisse plus y accéder. Le but est souvent de soutirer de l'argent en échange de la récupération de données, même si parfois, malgré le fait que la victime paye, la clef de déchiffrement ne sera jamais envoyée.

Le système de chiffrement utilisé est souvent symétrique car la vitesse de ce système est beaucoup plus rapide que le chiffrement asymétrique qui requiert deux clefs et qui est plus lent. Le chiffrement asymétrique est souvent utilisé pour envoyer la clef utilisée par le chiffrement symétrique afin qu'une tierce personne ne puisse la récupérer.

Le ransomware utilise aussi souvent le réseau, soit pour se propager, soit pour récupérer des informations (clef, données sensibles, ...) ce qui peut aussi être une faiblesse si l'adresse du serveur est retrouvée, ce qui permettrait d'identifier l'attaquant.

Fonctionnement global du ransomware



Le ransomware fonctionne comme ceci en ligne de commande : `./ransom <path> [key]`

- `path` est le chemin qui pointe vers le dossier ou le fichier à chiffrer
- `key` est la clef qui permet de déchiffrer un dossier ou un fichier

Compiler le programme

Il y a un **Makefile** qui permet de le compiler pour la plateforme de votre choix :

- `make linux` pour compiler pour Linux (avec *gcc*)
- `make windows` pour compiler pour Windows (avec *mingw*)
- `make server` pour compiler et lancer le serveur pour Linux (avec *gcc*)

Les fonctions du ransomware

Les fonctions principales sont :

> ***char* getKey()***
>
> `getKey()` *va essayer de se connecter au serveur de l'attaquant afin de recevoir une clef, si cela échoue, le programme va utiliser une clef qui est directement écrite dans le code (qui sera chiffré avec les autres fichiers lors du processus de chiffrement)*

> ***int processFile(char* path, const char* key)***
>
> `processFile(char *path, const char *key)` *reçoit le chemin du fichier à chiffrer ainsi que la clef. La fonction va regarder l'extension du fichier, si celui-ci ne se termine pas par* `.st` *alors le fichier sera chiffré*

> ***int encDir(char* path, const char* key)***
>
> `encDir(char* path, const char* key)` *est la fonction principale du programme, c'est elle qui va boucler de manière récursive afin d'envoyer en argument les fichiers à* `processFile()`.
> ***récursif*** *: _Qui peut être répété un nombre indéfini de fois par l'application de la même règle.*

Fonctions à usage plus basique

> `FILE* openFile(const char* path, const char* mode)`
> *ouvre un fichier dans le mode donné en argument et examine s'il y a des erreurs, retourne un pointeur vers la structure du fichier*

> `int closeFile(FILE*, FILE*)`
> *ferme le fichier d'entrée et de sortie*

> `int docrypt(FILE* input, FILE* output, const char* key,int (*) (int, char[], const char*,int*))`
> *reçoit un fichier d'entrée et de sortie et utilise la fonction donnée en argument pour soit chiffrer soit déchiffrer le fichier d'entrée*

> `int encrypt(int n, char[n], const char*, int*)`
> *chiffre la chaine de caractères donnée en argument*

> `int decrypt(int n, char[n], const char*, int*)`
> *déchiffre la chaine de caractères donnée en argument*

> `char* addext(const char* path, const char* ext)`
> *ajoute l'extension* `.st` *au fichier et renvoie le nom du fichier*

> `char* remext(const char* input)`
> *enlève l'extension* `.st` *au fichier et renvoie le nom du fichier*

> `int isDir(char* path)`
> *examine si le chemin donné pointe vers un dossier ou non*

> `char* addPath(const char* path, const char* file)`
> *ajoute le chemin d'accès complet au fichier et renvoie celui-ci à l'appelant*

> `void leaveExplanation()`
> *écrit dans* `stdout` *et dans un fichier* `readme` *comment récupérer les données.*

> `char* net_get(int* ID)`
> *reçoit un ID en argument et essaie de se connecter au serveur :*
> *s'il n'y arrive pas alors l'ID est mit à 0*
> *s'il y arrive alors renvoie la clef obtenue*

> `void send_ID(SOCKET sock, int* ID)`
> *envoie l'ID au serveur afin qu'il puisse générer une clef unique*

> `char* get_data(SOCKET sock)`
> *reçoit la clef du serveur*

> `SOCKET set_socket()`
> *crée le socket avec les bons paramètres et le renvoie*

> `SOCKADDR_IN set_addr()`
> *paramètre l'adresse ip et le port de destination du socket*

> `int bytes_to_hexa(const unsigned char bytes_string[], char *hex_string, int size)`
> *convertit une chaine de caractères en hexa*

> `int hexa_to_bytes(char hex_string[], unsigned char val[], int size);`
> *convertit un tableau de nombres hexa en chaine de caractères*

Les fonctions du serveur

> `char *gen_key(int ID)`
> *génère une clef en utilisant l'ID donné en paramètre et renvoie la clef générée*

> `void save(int ID, char* key, char* hkey)`
> *sauvegarde l'ID et la clef en ascii et en hexa dans un fichier*

> `void handleClients(SOCKET sock)`
> *s'occupe d'un client (recevoir l'ID, générer la clef, envoyer la clef)*

Exemple

- Voici un exemple d'arborescence quelconque

```
data/
├── 0003.JPG
├── sub1
|   ├── parent.exe
|   ├── sub11
|   |   ├── 81Z67s4AKCL.jpg
|   |   └── main.c
|   └── trollware2.spec
└── sub2
    ├── 0003.JPG
    ├── sub21
    |   ├── 0003.JPG
    |   └── server
    └── sub22
        ├── Featured.jpg
        └── keylog-master.zip

5 directories, 10 files
```

- Lancement du programme afin de chiffrer ce dossier

```
> ./bin/r.lin data

-> data/0003.JPG
processing : data/0003.JPG.st
-> data/sub1
-> data/sub1/parent.exe
processing : data/sub1/parent.exe.st
```

```
-> data/sub1/sub11
-> data/sub1/sub11/81Z67s4AKCL.jpg
processing : data/sub1/sub11/81Z67s4AKCL.jpg.st
-> data/sub1/sub11/main.c
processing : data/sub1/sub11/main.c.st
-> data/sub1/trollware2.spec
processing : data/sub1/trollware2.spec.st
-> data/sub2
-> data/sub2/0003.JPG
processing : data/sub2/0003.JPG.st
-> data/sub2/sub21
-> data/sub2/sub21/0003.JPG
processing : data/sub2/sub21/0003.JPG.st
-> data/sub2/sub21/server
processing : data/sub2/sub21/server.st
-> data/sub2/sub22
-> data/sub2/sub22/Featured.jpg
processing : data/sub2/sub22/Featured.jpg.st
-> data/sub2/sub22/keylog-master.zip
processing : data/sub2/sub22/keylog-master.zip.st
Hello,
Your files are now encrypted with the extension .st
If you wanna recover your files you have to send us
your ID
Only then we can start talking for the price^^
Have a good day
Your ID is : 0
```

- Arborescence post chiffrement

```
data
├── 0003.JPG.st
├── sub1
|   ├── parent.exe.st
|   ├── sub11
|   |   ├── 81Z67s4AKCL.jpg.st
|   |   └── main.c.st
|   └── trollware2.spec.st
└── sub2
    ├── 0003.JPG.st
    ├── sub21
    |   ├── 0003.JPG.st
    |   └── server.st
```

```
    └── sub22
        ├── Featured.jpg.st
        └── keylog-master.zip.st

5 directories, 10 files
```

- Lancement du programme afin de déchiffrer ce dossier

```
> ./bin/r.lin data
717d7d63292c6170783f69632a6e7d7a797b2c757629292c24706
6686a783a
-> data/0003.JPG.st
processing : data/0003.JPG
-> data/sub1
-> data/sub1/parent.exe.st
processing : data/sub1/parent.exe
-> data/sub1/sub11
-> data/sub1/sub11/81Z67s4AKCL.jpg.st
processing : data/sub1/sub11/81Z67s4AKCL.jpg
-> data/sub1/sub11/main.c.st
processing : data/sub1/sub11/main.c
-> data/sub1/trollware2.spec.st
processing : data/sub1/trollware2.spec
-> data/sub2
-> data/sub2/0003.JPG.st
processing : data/sub2/0003.JPG
-> data/sub2/sub21
-> data/sub2/sub21/0003.JPG.st
processing : data/sub2/sub21/0003.JPG
-> data/sub2/sub21/server.st
processing : data/sub2/sub21/server
-> data/sub2/sub22
-> data/sub2/sub22/Featured.jpg.st
processing : data/sub2/sub22/Featured.jpg
-> data/sub2/sub22/keylog-master.zip.st
processing : data/sub2/sub22/keylog-master.zip
Your files are now decrypted !
```

- Arborescence post déchiffrement

```
data
├── 0003.JPG
```

```
├── sub1
|   ├── parent.exe
|   ├── sub11
|   |   ├── 81Z67s4AKCL.jpg
|   |   └── main.c
|   └── trollware2.spec
└── sub2
    ├── 0003.JPG
    ├── sub21
    |   ├── 0003.JPG
    |   └── server
    └── sub22
        ├── Featured.jpg
        └── keylog-master.zip

5 directories, 10 files
```

Conclusion

En conclusion, il est préférable de ne pas exécuter un programme sans être sûr de la provenance de celui-ci.
Car en développant ce projet, surtout en faisant des tests, on se rend rapidement compte que sans la clef de déchiffrement, il est très difficile de pouvoir récupérer les fichiers intacts.
En cas de manipulation distraite, une solution de rattrapage ne serait pas simple. Si malgré le versement d'argent, la clef n'est pas correcte, voire inexistante, une solution de dernier espoir serait d'essayer de récupérer un fichier à la force brute (essayer toutes les combinaisons possibles). Des backups sont donc fortement conseillés !