TP : Obfuscation d'un programmes Python

Nadhir Boukhechem

Les codes malveillants utilisent l'obfuscation pour masquer leur comportement réel, ralentir l'analyse statique et contourner des mécanismes de détection. Dans ce TP, vous allez obfusquer progressivement un programme Python simple sans jamais modifier son comportement fonctionnel.

Voici un petit programme Python de vérification de mot de passe :

```
def verifier_mot_de_passe():
    mot_de_passe = "utec2026"
    saisie = input("Entrez le mot de passe : ")

    if saisie == mot_de_passe:
        print("Accès autorisé")
    else:
        print("Accès refusé")

verifier_mot_de_passe()
```

Q1) Pourquoi ce code est-il facile à analyser ? Quelle est la partie la plus intéressante à masquer ?

Ce code est facile à analyser tout d'abord par le fait que le code soit en clair.
Secondement, la non compléxité de code rend également l'analyse facile.
La partie la plus intéressante à masquer serait la variable 'mot_de_passe', notamment son contenu.

Obfuscation lexicale :

Q2) Modifier le code afin de :

- Renommer les variables et fonctions avec des noms non significatifs
- Supprimer toute information facilitant la lecture
- Conserver exactement le même comportement à l'exécution

```
def null_a():
    a8 = "utec2026"
    b98 = input("Entrez le mot de passe : ")
    if a8 == b98:
        print("Accès autorisé")
    else:
        print("Accès refusé")

null_a()
```

Obfuscation du flot de contrôle :

Q3) Vous devez rendre la logique du programme plus difficile à suivre en utilisant :

- Des conditions inutiles
- Des variables intermédiaires
- Des structures de contrôle redondantes
- Un ordre d'exécution peu intuitif

```
def null_a():
    a8 = "utec2026"
    b98 = input("Entrez le mot de passe : ")
    c6=(b98==a8)

    if c6==c6:
        if c6!=False:
            print("Accès autorisé")
        else:
            if c6!=True:
                print("Accès refusé")

if __name__==”__main__”:
    null_a()
```

Q4) Quel est l'impact sur les performances ?

Ici, il n'y a aucun impact, mais selon certains ajouts celà peut ralentir son exécution. On peut en déduire le non impact de par la complexité du programme en O(1).

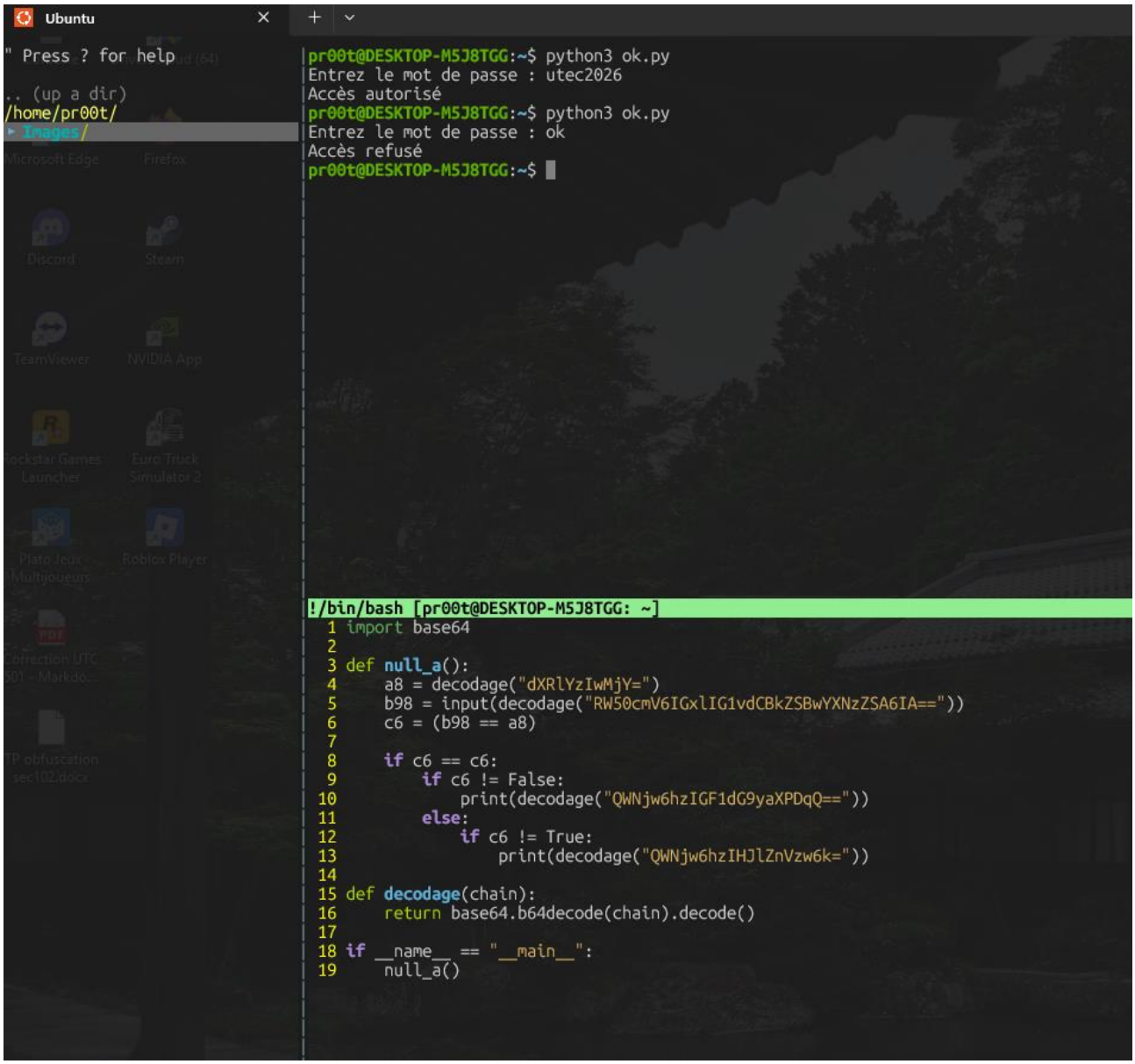
Obfuscation des données :

Q5) Les chaînes de caractères sensibles ne doivent plus apparaître en clair dans le code. Vous devez :

- Masquer le mot de passe
- Masquer les messages affichés
- Reconstruire les données dynamiquement à l'exécution

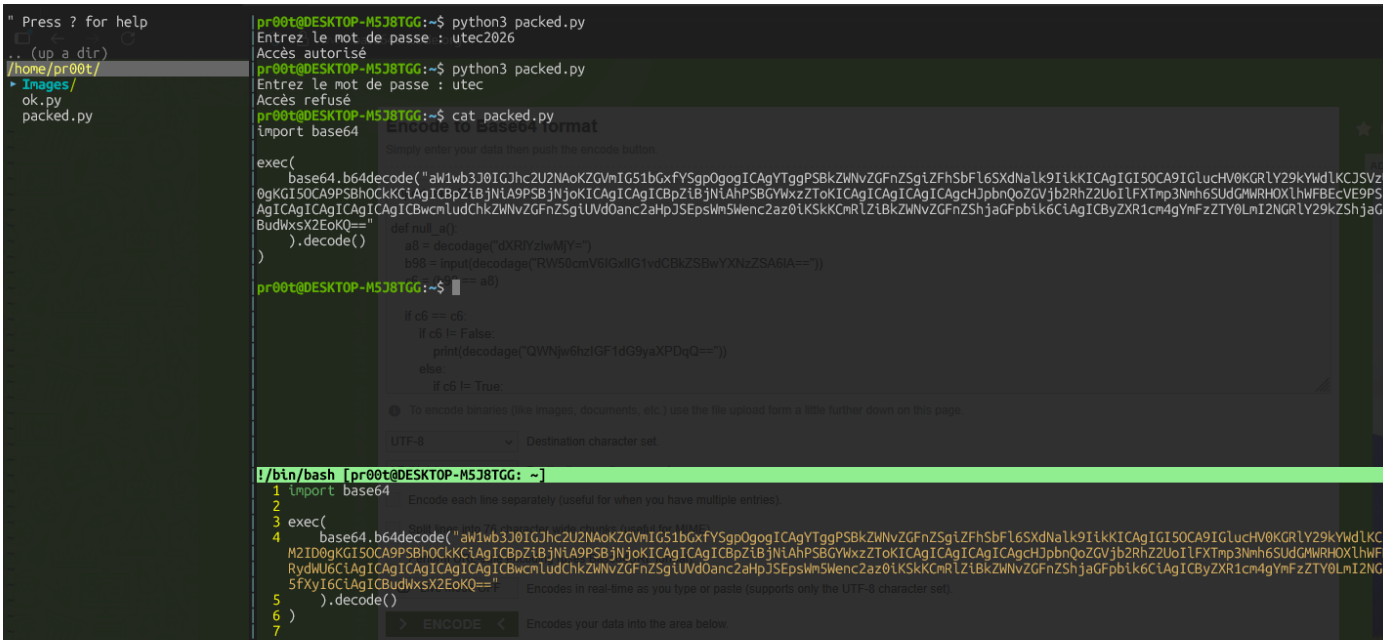
Vous pouvez utiliser différentes techniques :

- Encodez les chaînes de caractères (Base64, Hexadécimal, ou décalage César).
- Créez une fonction de "décodage" qui reconstruit la chaîne uniquement au moment de la comparaison.



Packing :

Q6) Mettre en œuvre un mécanisme de packing en Python en utilisant la fonction exec() appliquée à une chaîne de caractères compressée ou encodée, afin de dissimuler l'ensemble du script original.



Q7) Citer et présenter brièvement quelques packers existants sur le marché, utilisés pour la protection ou l'obfuscation des programmes.

UPX est un packer open-source qui compresse les exécutables pour réduire leur taille et rendre plus complexe mais en gardant le programme inital fonctionnel.

PyArmor est un outil Python qui obfusque les scripts en rendant le script illisible et un ajoute un loader, ce qui permet d'exécuter en décryptant le code en mémoire sans jamais exposer le code source en clair sur le disque.

Q8) Expliquez le rôle des débogueurs dans l'analyse de programmes obfusqués.

Le rôle d'un débogueur est d'éxecuter un programme pas à pas afin d'étudié son comportement, l'état des variables à chaque instant, grâce à l'utilisation de breakpoints et autre..

Malgré un programme obfusqué, le debogueur donne le comportement réel du programme, par son exécution.

Celà permet également d'obtenir des informations de protections ou de mécanismes anti-debug.

Exemple avec un binaire provenant d'un challenge de la plateforme root-me.org:

```
gdb ./ch3.bin
run
Starting program: chall/root-me/ch3.bin
Debugger detecté ... Exit
[Inferior 1 (process 3444) exited with code 01]
```

Q9) Comment les codes malveillants cherchent-ils à détecter ou à bloquer l'utilisation des débogueurs ?

Les codes malveillants détectent les debuggers en verifiant l'environnement dans lequel il est exécuté et bloquent leur fonctionnement en modifiant ou arrêtant leur comportement
Les debuggers peuvent determiner leur environnement par la vérification de flags sytème, detection de breakpoints..