1 NOTES IMPORTANTES

Le rendu final doit avoir l'arborescence ci-dessous. Respecter impérativement la nomenclature des répertoires, fichiers et fonction. Les fichiers/repertoires supplémentaires ne seront pas pris en compte, les inclure ne vous pénalise pas.
ATTENTION : NE PAS INCLURE LE MODELE LLM

```
RENDU_<ID ELEVE>
├─ email/
│   ├─ mail_raw.txt
│   ├─ gen_mail.py
│   └─ test_mail/
├─ page/
│   ├─ app.py
│   ├─ templates/
│   │   └─ chrono_clone.html
│   └─ static/
│      ├─ style.css
│      ├─ script.js
│      └─ images/
│         └─ logo.png
│       …
├─ automat/
│   ├─ model/
│   ├─ flask_server/
│   ├─ test_mail/
│   └─ auto_attack.py
└─ cibles.csv
```

```
# Le répertoire static contiendra tous les fichiers relatifs au bon fonctionnement de la page
chrono_clone.html (section 2 question 3). Les fichiers qui sont dans cette arborescence sont
un exemple et peuvent ne pas avoir les mêmes noms que vos fichiers
```

2 Introduction

Le but du TP est de créer une campagne de Phishing. Les éléments nécessaires au succès de cette campagne sont :

- Clone de la page de Chronopost qui invite la cible à reprogrammer la livraison à laquelle on ajoutera un champ pour entrer les détails d'une carte bancaire
- Clone d'un mail Chronopost dont tous les liens pointeront vers la page de Phishing
- Une liste de cible : Nom, Prénom, adresse, email
- Un script qui envoie les messages aux cibles
- Automatisation des attaques

3 Le mail

Le but est de transformer un mail légitime Chronoposte. Pour ce faire, prendre un mail légitime reçu dans votre boite mail (version raw HTML) et remplacer tous les

liens par un lien qui pointe vers la page de phishing en remplaçant les données pertinentes par celle de l'utilisateur. Fichier : gen_mail.py

1. Écrire une fonction re_encode qui prend en paramètre le chemin d'un fichier, l'ouvre en mode binaire, le lit, décode les données (iso-8859-1) et sauvegarde le résultat dans le même chemin en remplaçant l'extension quel qu'elle soit par html (ex : .txt > .html)
2. Écrire une fonction fix_links qui prend en paramètre le chemin d'un fichier, le lit, remplace les liens par le lien de la page de phishing en incluant comme paramètre HTML commande=<*numéro de la commande client*>, et sauvegarde le résultat dans le même fichier.
3. Sachant que le mail orignal était destiné à Dupond et Dupont - 82 ALLEE DES FLEURS ETAGE 1 – 96969 VALHALLA, commande numéro FR123456789AB, remplacer les valeurs statiques par des variables compatibles avec Jinja2.
4. Écrire une fonction generate_email qui prend en paramètre le chemin du fichier mail et un dictionnaire qui contient les coordonnées du destinataire, lit le contenu du fichier, compile la template en utilisant les coordonnées du destinataire et retourne le résultat. Ne pas oublier les liens vers la page de Phishing : le numéro de la commande doit aussi être lu à partir des coordonnées client.
5. Écrire une fonction qui prend en paramètre le chemin d'un fichier CSV contenant les cibles avec leurs coordonnées et le chemin du fichier template, lit le fichier CSV et génère des mails de phishing pour chaque cible. Simuler l'envoi par mail en sauvegardant le résultat de chaque message dans un fichier NOM_DE_LA_CIBLE.HTML dans le répertoire test_mail.

4 Page de phishing

Un serveur flask est un serveur web léger en python utilisé pour servir des pages HTML. Il sera utilisé comme serveur de la page de phishing.

1. En se basant sur la page chrono/chronopost_page.html et les fichiers/répertoires adjacents, et en examinant la page de test existante, répartir les fichiers adéquatement dans l'arborescence flask pour que la page de phishing soit activée
2. Modifier le code dans app.py pour le pointer vers la nouvelle page quand l'utilisateur essaye d'accéder au répertoire racine (ex : http://localhost/)
3. Modifier la page html pour pointer les images/css/js vers les bons fichiers (examiner la page de test pour comprendre comment faire les liens)
4. Ajouter un champ obligatoire dans la section « Modifier la date de livraison » dans lequel l'utilisateur doit ajouter sa carte de crédit avec la mention « coût de l'opération : 1 euro ».
5. Remplacer, dans la page html, les valeurs statiques de Dupond et Dupont par les coordonnées clients (dynamique). Flask utilise Jinja2 pour les templates.
6. Ajouter à app.py une fonction get_client qui prend en paramètre un numéro de commande, lit un fichier CSV prédéfini, et retourne les coordonnées du client concerné

7. Modifier la ligne où flask compile la page pour lui passer les coordonnées client en paramètre
8. Tester la page en passant un numéro de commande valide
9. Modifier le code relatif à la requête POST pour extraire le numéro de la carte de crédit et le sauvegarder dans un fichier ainsi que le numéro de la commande et les autres coordonnées pertinentes
10. Test la requête POST

5 Automatisation

Cette étape consiste à automatiser la création de page de Phishing et l'envoie de mail en utilisant les LLM

1. En utilisant vos connaissances (requests, mechanize, bs4, …) écrire une fonction qui prend en paramètre un lien et :
    a. Télécharge la page du lien ainsi que toutes les images/js/css
    b. Remplace tous les liens de la page par des liens valides vers les images/js/css téléchargés (utiliser bs4 ?)
    c. Recherche un formulaire (n'importe lequel) et y ajoute un champ obligatoire qui demande l'ajout d'une carte de crédit
    d. Sauvegarde le tout (page modifiée et images/js/css) dans une structure adéquate utilisable par flask dans le répertoire flask_server.
2. En utilisant les exos de LLM, écrire une fonction qui prend en paramètre un chemin de fichier CSV :
    a. Lit le fichier (coordonnées cibles)
    b. Génère des mails de phishing qui pointent vers la page clonée et les sauvegardent un à un dans un répertoire test_mail.