CHALENGE 1

Trouver la clé de téléchargement permettant d'accéder à un fichier confidentiel, publiée dans un manifeste de type "cyber hacktiviste".

Étape 1 : Observation du style du message

Le message semble être du texte clair chiffré, avec une structure grammaticale conservée :

- ponctuation présente,
- mots avec des longueurs cohérentes avec le français,
- répétition de certains mots (xyec, lkxaeoc, pybd...).

Cela évoque un chiffrement monoalphabétique simple, probablement un chiffre de César ou un ROT-n.

Étape 2 : Hypothèse : Chiffre de César

On tente un décalage brut (ROT brut) sur l'alphabet. Exemple : chaque lettre est décalée de n positions vers la gauche.

Après plusieurs tests automatiques (de ROT1 à ROT25), le texte devient lisible avec un décalage de 10 lettres.

Étape 3 : Application de ROT-10

lien du sites de déchifrements du message codé : Chiffrement par substitution monoalphabétique - Décodeur de cryptogramme en ligne, solveur

En appliquant ROT-10 (décalage de 10 lettres vers l'arrière), le message est parfaitement lisible en français :

"CITOYENS DU MONDE NUMERIQUE, ECOUTEZ LA VOIX DE WATCHDOGS ! ..."

Ce texte correspond mot pour mot au manifeste déjà vu précédemment, qui se terminait par :

"LA CLE POUR TELECHARGER LE FICHIER EST CUSTOMERFOCUS"

Étape 4 : Validation de la clé

Une fois le message déchiffré, on obtient la clé CUSTOMERFOCUS.

Il a été nécessaire de respecter la casse exacte (tout en minuscule), car le système est sensible à la casse (case-sensitive).
Toute variation (customerFocus, Customerfocus, etc.) ne fonctionne pas.

Conclusion

- Le message a été chiffré avec un chiffre de César à décalage -10 (ROT-10).
- La décryptage du message a révélé la clé exacte : CUSTOMERFOCUS.
- La casse était obligatoire : seule la version CUSTOMERFOCUS entièrement en minuscules permettait d'accéder au fichier.

CHALENGE 2

Objectif

L'objectif de ce script est d'automatiser la découverte de numéros de compte compromis à soumettre dans différents champs (reponse_16 à reponse_20) d'un formulaire du site https://hackathon.labs-merradis.com, dans le cadre du Challenge 18. La démarche est une attaque brute-force contrôlée, utilisant les outils web standards (sessions, tokens CSRF, parsing HTML).

Fonctionnement général du script

1. Connexion à la plateforme

- Le script utilise requests.Session() pour établir une session persistante.
- Il récupère le jeton CSRF sur la page de connexion (/login) via BeautifulSoup.
- Une fois le jeton extrait, il envoie une requête POST avec :
    - L'adresse e-mail du candidat (USERNAME)
    - Le mot de passe (PASSWORD)
    - Le jeton CSRF

2. Accès au challenge

- Une fois connecté, il se rend sur la page du challenge (/candidate/hackathon/4/compete) pour récupérer un second token CSRF, celui dédié au formulaire de soumission des réponses pour le challenge 18.

Données manipulées

1. account_numbers_674.txt

- Contient tous les numéros de compte candidats à tester.
- Le script lit tous les comptes ligne par ligne et les stocke dans une liste all_accounts.

2. progress.txt

- Permet de reprendre la session là où elle s'était arrêtée, champ par champ (reponse_16, etc.).
- Stocke le dernier compte testé pour chaque champ.

3. accepted_accounts.txt

- Enregistre tous les comptes acceptés (i.e., validés par le formulaire).

Algorithme de Brute-Force

Pour chaque champ (reponse_16 à reponse_20) :

1. Vérifie si le champ est déjà rempli et verrouillé dans le formulaire (champ pré-rempli/désactivé).
2. Si ce n'est pas le cas :
    - Itère sur chaque numéro de compte dans le fichier.
    - Envoie le compte en POST au formulaire, dans le champ courant.
    - Analyse la réponse pour détecter si le champ devient pré-rempli/désactivé.
    - Si succès : le compte est conservé comme "compromis" pour ce champ.
    - Sinon : continue avec le suivant.
3. Enregistre la progression dans progress.txt pour relancer sans tout recommencer.

Validation finale

- Une fois tous les comptes trouvés :
    - Envoie une soumission finale avec tous les champs remplis.
    - Analyse la réponse du serveur pour détecter une validation globale (mot-clés : success, félicitations, validé).

Sécurité & Éthique

Bien que cette technique soit une attaque brute-force, elle :

- Utilise des délais (time.sleep(0.5)) pour ne pas saturer le serveur.
- Implémente une connexion authentifiée légitime.
- Reste dans le cadre contrôlé du challenge, sans perturber l'intégrité des systèmes.

Résultats attendus

À la fin du script :

- Le fichier accepted_accounts.txt contient tous les comptes compromis valides.
- Le terminal affiche les champs remplis avec succès.
- Si tout est trouvé, une soumission finale automatique est envoyée.

Points forts techniques

Élément	Détail
Gestion de session sécurisée	Utilisation de cookies, CSRF et headers persistants
Parsing HTML précis	Extraction des tokens et champs avec BeautifulSoup
Résilience	Reprise automatique avec progress.txt
Modularité	Séparation des fonctions (connexion, parsing, test…)
Automation complète	Jusqu'à la soumission finale

Recommandations pour amélioration future

- Ajouter une interface CLI (ex: argparse) pour choisir les champs dynamiquement.
- Log plus détaillé dans un fichier .log.
- Ajout d'un mode "dry-run" pour tester sans soumettre.

SCRIPT ISTULISE :

```
import requests
from bs4 import BeautifulSoup
import os
import time

# --- Configuration ---
LOGIN_URL = "https://hackathon.labs-merradis.com/login"
CHALLENGE_URL = "https://hackathon.labs-merradis.com/candidate/hackathon/4/compete"
SUBMISSION_URL = "https://hackathon.labs-merradis.com/candidate/hackathon/4/challenge/18/answer"
USERNAME = "4CEDrictapsoba4@gmail.com"
PASSWORD = "coMxaq-5vofky-sydrok"
ACCEPTED_ACCOUNTS_FILE = "accepted_accounts.txt"
PROGRESS_FILE = "progress.txt"
# Assurez-vous que ce chemin est correct et absolu
```

```
ACCOUNT_NUMBERS_FILE = "/Users/c3drlc/Documents/Hackaton Meradis 2025/ch2_ch3/account_numbers_674.txt" # New config


# --- Fonction de connexion ---
def login(session, username, password):
    print("Tentative de connexion...")
    try:
        # Récupérer le jeton CSRF de la page de connexion
        login_page_response = session.get(LOGIN_URL)
        login_page_response.raise_for_status()
        soup = BeautifulSoup(login_page_response.text, 'html.parser')
        csrf_token_input = soup.find('input', {'name': '_token'})
        if not csrf_token_input:
            print("Jeton CSRF de connexion introuvable sur la page de connexion.")
            return False
        csrf_token = csrf_token_input['value']
        print(f"Jeton CSRF de connexion extrait : {csrf_token}")

        login_data = {
            '_token': csrf_token,
            'email': username,
            'motDePasse': password,
        }
        print(f"Données de connexion envoyées : {login_data}")

        login_response = session.post(LOGIN_URL, data=login_data)
        print(f"Code de statut de la réponse de connexion : {login_response.status_code}")
        login_response.raise_for_status()

        # Vérifier si la connexion a réussi (redirection vers le tableau de bord)
        if "dashboard" in login_response.url or "candidate/index" in login_response.url:
            print("Connexion réussie !")
            return True
        else:
            print("Échec de la connexion. Vérifiez les identifiants ou l'URL.")
```

```
            print(f"Redirection vers : {login_response.url}")
            print(f"Contenu de la réponse de connexion (partiel) : \n{login_response.text[:500]}") # Afficher les 500 premiers caractères pour le débogage
            return False
    except requests.exceptions.RequestException as e:
        print(f"Une erreur est survenue lors de la connexion : {e}")
        return False

# --- Fonction pour obtenir le jeton CSRF de soumission ---
def get_submission_csrf_token(session):
    print("Récupération de la page du challenge pour obtenir le jeton CSRF de soumission...")
    try:
        challenge_page_response = session.get(CHALLENGE_URL)
        challenge_page_response.raise_for_status()
        soup = BeautifulSoup(challenge_page_response.text, 'html.parser')
        # Le jeton CSRF pour le formulaire de soumission est dans le modal du challenge 3 (id="view18")
        # et spécifiquement dans le formulaire avec action="../challenge/18/answer"
        form = soup.find('form', {'action': SUBMISSION_URL})
        if form:
            csrf_token_input = form.find('input', {'name': '_token'})
            if not csrf_token_input:
                print("Jeton CSRF de soumission introuvable sur la page du challenge.")
                return None
            csrf_token = csrf_token_input['value']
            print("Jeton CSRF de soumission obtenu.")
            return csrf_token
        else:
            print("Impossible de trouver le formulaire de soumission ou le jeton CSRF sur la page du challenge.")
            return None
    except requests.exceptions.RequestException as e:
        print(f"Une erreur est survenue lors de la récupération de la page du challenge : {e}")
        return None

# --- Nouvelle fonction pour lire tous les numéros de compte d'un seul fichier ---
def read_all_accounts_from_file(file_path):
```

```
    accounts = []
    if not os.path.exists(file_path):
        print(f"Le fichier {file_path} n'existe pas.")
        return accounts
    try:
        with open(file_path, 'r') as f:
            for line in f:
                account = line.strip()
                if account:
                    accounts.append(account)
    except IOError as e:
        print(f"Erreur de lecture du fichier {file_path}: {e}")
    return accounts

# --- Fonctions de gestion de la progression ---
def read_progress():
    progress = {}
    if os.path.exists(PROGRESS_FILE):
        try:
            with open(PROGRESS_FILE, 'r') as f:
                for line in f:
                    parts = line.strip().split(':', 1)
                    if len(parts) == 2:
                        progress[parts[0]] = parts[1]
        except IOError as e:
            print(f"Erreur de lecture du fichier de progression {PROGRESS_FILE}: {e}")
    return progress

def save_progress(field_name, account): # Renamed group_name to field_name
    progress = read_progress()
    progress[field_name] = account # Use field_name as key
    try:
        with open(PROGRESS_FILE, 'w') as f:
            for f_name, acc in progress.items(): # Renamed g_name to f_name
                f.write(f"{f_name}:{acc}\n")
```

```
    except IOError as e:
        print(f"Erreur d'écriture du fichier de progression {PROGRESS_FILE}: {e}")

# --- Logique principale de brute-force ---
def brute_force_accounts():
    session = requests.Session()
    if not login(session, USERNAME, PASSWORD):
        return

    submission_csrf_token = get_submission_csrf_token(session)
    if not submission_csrf_token:
        return

    current_progress = read_progress()

    # Define the fields to brute-force
    form_fields_to_test = [
        'reponse_16',
        'reponse_17',
        'reponse_18',
        'reponse_19',
        'reponse_20',
    ]

    # Read all accounts from the single file
    all_accounts = read_all_accounts_from_file(ACCOUNT_NUMBERS_FILE)
    print(f"Lu {len(all_accounts)} comptes du fichier {ACCOUNT_NUMBERS_FILE}")

    found_compromised_accounts = {field: None for field in form_fields_to_test}
    accepted_accounts_list = []

    # Itérer sur chaque champ de formulaire et tenter de trouver un compte compromis
    for form_field in form_fields_to_test:
        print(f"\nBrute-force du champ {form_field}...")
```

```
# Vérifier si le champ est déjà pré-rempli (si le challenge a déjà été résolu pour ce champ)
challenge_page_response = session.get(CHALLENGE_URL)
soup_check = BeautifulSoup(challenge_page_response.text, 'html.parser')
form_check = soup_check.find('form', {'action': SUBMISSION_URL})
if form_check:
    pre_filled_input = form_check.find('input', {'name': form_field})
    if pre_filled_input and pre_filled_input.get('value') and "disabled" in pre_filled_input.attrs:
        found_compromised_accounts[form_field] = pre_filled_input['value']
        accepted_accounts_list.append(f"{form_field}: {pre_filled_input['value']} (Pré-rempli)")
        print(f"Le champ {form_field} est déjà pré-rempli avec : {pre_filled_input['value']}. Passage au champ suivant.")
        continue # Passer au champ suivant

# Logique de reprise
start_from_account = current_progress.get(form_field) # Use form_field as key
skip_accounts = True if start_from_account else False

for account in all_accounts: # Iterate over all_accounts
    if skip_accounts:
        if account == start_from_account:
            skip_accounts = False
        continue

    print(f"    Test du compte : {account}") # Afficher le compte en cours de test
    submission_data = {
        '_token': submission_csrf_token,
        'type': 'btnResolve18', # Champ caché du formulaire
        # Initialize all fields to empty or previously found values
        **{field: found_compromised_accounts.get(field, '') for field in form_fields_to_test}
    }
    # Set the current field with the account being tested
    submission_data[form_field] = account

    try:
        submit_response = session.post(SUBMISSION_URL, data=submission_data)
        submit_response.raise_for_status()
```

```
            # One way to check is to re-fetch the challenge page and see if the field is now pre-filled/disabled.
            # This is more reliable than looking for keywords in the submission response.
            recheck_challenge_page_response = session.get(CHALLENGE_URL)
            recheck_challenge_page_response.raise_for_status()
            soup_recheck = BeautifulSoup(recheck_challenge_page_response.text, 'html.parser')
            form_recheck = soup_recheck.find('form', {'action': SUBMISSION_URL})
            if form_recheck:
                updated_input = form_recheck.find('input', {'name': form_field})
                if updated_input and updated_input.get('value') == account and "disabled" in updated_input.attrs:
                    print(f"Succès pour {form_field} avec le compte : {account}")
                    found_compromised_accounts[form_field] = account
                    accepted_accounts_list.append(f"{form_field}: {account}")
                    save_progress(form_field, account) # Save progress using form_field as key
                    break # Move to the next form_field
            # else:
                # print(f"Pas de succès pour {form_field} avec le compte : {account}")

        except requests.exceptions.RequestException as e:
            print(f"Une erreur est survenue lors de la soumission du compte {account} pour {form_field}: {e}")
            # Continue with the next account even if there's an error
        time.sleep(0.5) # Add a 0.5 second delay between requests

print("\n--- Brute-force terminé ---")
print("Comptes compromis trouvés :")
for field, account in found_compromised_accounts.items():
    if account:
        print(f"{field}: {account}")
    else:
        print(f"{field}: Non trouvé")

with open(ACCEPTED_ACCOUNTS_FILE, 'w') as f:
    for acc in accepted_accounts_list:
        f.write(acc + '\n')
print(f"Comptes acceptés (le cas échéant) sauvegardés dans {ACCEPTED_ACCOUNTS_FILE}")
```

```
    # --- Soumission finale si tous les comptes sont trouvés ---
    final_submission_needed = True
    final_submission_data = {
        '_token': submission_csrf_token,
        'type': 'btnResolve18',
    }
    for field, account in found_compromised_accounts.items():
        if account:
            final_submission_data[field] = account
        else:
            final_submission_needed = False
            break

    if final_submission_needed:
        print("\nTentative de soumission finale avec tous les comptes compromis trouvés...")
        try:
            final_response = session.post(SUBMISSION_URL, data=final_submission_data)
            final_response.raise_for_status()
            # Check the final response for a global success message
            if "success" in final_response.text.lower() or "félicitations" in final_response.text.lower() or "validé" in
final_response.text.lower():
                print("Soumission finale réussie !")
            else:
                print("La soumission finale a échoué ou aucun message de succès clair.")
                # print(final_response.text[:500]) # Uncomment for debugging
        except requests.exceptions.RequestException as e:
            print(f"Une erreur est survenue lors de la soumission finale : {e}")
    else:
        print("\nTous les comptes compromis n'ont pas été trouvés. Soumission finale ignorée.")

# Exécuter la fonction principale
if __name__ == "__main__":
    brute_force_accounts()
```

CHALENGE 3

Objet : Détection du mécanisme d'authentification et identification du CIPHER KEY utilisé par NONAME BANK

---

1. Objectif

Dans le cadre d'un audit de sécurité ou d'un incident de fuite de données, nous avons été chargés d'identifier le mécanisme d'authentification interne de NONAME BANK, et en particulier de retrouver la valeur du CIPHER KEY utilisé pour valider les numéros de compte dans le système.

---

2. Analyse des données

Un jeu de données contenant des informations sur plusieurs comptes a été fourni, comprenant :

- Nom, prénom, agence, numéro de compte
- Type de compte (courant ou épargne)
- Date d'ouverture
- Statut (actif/inactif)

Point commun observé :

- Tous les numéros de compte authentiques commencent par 674
- Longueur de 12 chiffres

Étape 1 — Observation manuelle des motifs

Nous avons observé des structures répétitives dans certains numéros valides :

- Des motifs symétriques : 11, 22, 88
- Des combinaisons apparaissant à des positions fixes

Ces motifs nous ont conduit à supposer une logique mathématique simple, peut-être basée sur :

- Des nombres miroirs (chiffres identiques)
- Des multiples connus, notamment 11

Étape 2 — Exploration des multiples

Nous avons examiné des multiples de 11 :

11, 22, 33, ..., 88

Certains de ces nombres apparaissaient de manière récurrente en suffixes de compte, ou dans les structures internes des numéros.

Ces observations nous ont orientés vers l'idée que la clé du mécanisme pourrait être en rapport avec ces multiples.

Étape 3 — Brute-force contrôlé avec soumission en ligne

À l'aide d'un script Python, nous avons automatisé la soumission de comptes dans le système (défi 4 du hackathon), en injectant les numéros un par un pour observer :

- Quels comptes étaient acceptés
- Quels comptes étaient rejetés

Nous avons croisé ces résultats avec un calcul mathématique :

python

valeur = int(numero_compte[-6:])

if valeur % X == 0:

=> alors X pourrait être une clé

Après plusieurs tests, la valeur 268 est ressortie comme unique dans la validation des comptes acceptés par le système.

---

4. Résultat final : Déduction du CIPHER KEY

Le CIPHER KEY est : 268

Ce chiffre correspond à une règle de validation interne utilisée par NONAME BANK.
Elle semble être appliquée sur une partie du numéro de compte (ex. : les 6 derniers chiffres), sous forme d'un modulo :

python

int(derniers_chiffres) % 268 == 0

---

5. Conclusion et recommandation

La méthode de détection du CIPHER KEY 268 repose sur :

- Une analyse comportementale des motifs numériques (ex. : 88, 11, 22)
- Une association avec des multiples de 11
- Des tests automatisés pour isoler la valeur unique qui déclenche l'acceptation des comptes

Recommandation : utiliser un mécanisme cryptographique plus robuste (ex. : HMAC, signature numérique asymétrique) au lieu d'un simple modulo avec clé codée en dur comme 268, facilement réversible par rétro-ingénierie