

AI CTF – Write up

HeadMind Partners
BELGIUM

FOR A SECURE AND RESPONSIBLE
DIGITAL SOCIETY





Implementation error

Challenge inspiré du TRACS 2024

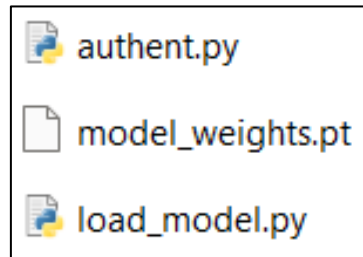
Contexte

Une organisation secrète veut contrôler ses armes à l'aide d'une intelligence artificielle ultra-sécurisée. Seul un QR code valide peut désactiver leurs dispositifs. Leur système repose sur un modèle d'IA qui analyse ces QR codes pour vérifier leur authenticité.

Mais leur IA n'est pas si parfaite...

Votre mission : trompez leur système en trouvant une image capable de passer l'authentification. Si vous réussissez, vous obtiendrez le code de désactivation. ✨

Trois fichiers sont fournis pour ce challenge :



Authent.py vérifie la validité du code QR

Model_weights.pt représente le réseau de neuronne

Load_model.py est un utilitaire pour charger et executer l'IA

Solution

Seul `authent.py` est utile pour ce challenge.

```
from load_model import load_model

MODEL = load_model("model_weights.pt")
MODEL.eval() #On passe l'IA en mode utilisation
```

Le modèle est chargé via l'utilitaire `load_model.py`

```
def authentication(image):
```

```
    if image.shape != (1, 32, 32):
        raise ValueError("L'image n'est pas au format attendu (1x32x32).")
```

Le format de l'image est vérifiée (Canal, Hauteur, Largeur)

```
    score = MODEL(image)[0, 1].item()
```

L'IA donne **un score de 0 à 100 %** sur la validité du QR code.

```
    if score < 1.1:
```

```
        return "❌ Score insuffisant."
```

```
    return "🚩 Flag : HMP{fake_flag}"
```

Si le **score est d'au moins 110%**, alors le QR est valide

Solution

Il est normalement impossible d'atteindre un score aussi élevé (le seuil est fixé à $\text{score} < 1.1$, soit 110 % de "certitude"). Cela rend le challenge infranchissable... sauf si on abuse d'une erreur logique dans le code de vérification.

Le modèle d'IA retourne un score que l'on compare avec 1.1. Mais ce score peut aussi être NaN (Not a Number) si l'entrée est malformée. Or `nan < 1.1 → False`

Il suffit alors de faire une image au format valide mais aux valeurs de pixels invalides (nan, inf...)

```
image = torch.full(size: (1, 32, 32), float('nan'))
```

Finalement ...

```
HMP{nan_bypass_due_to_bad_score_check}
```

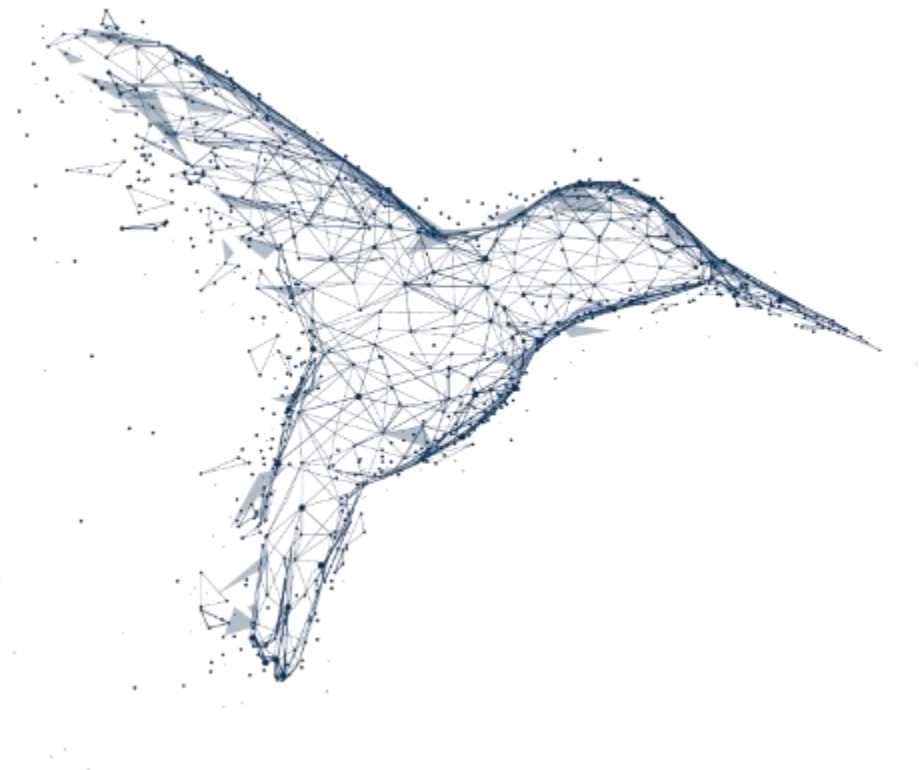
Remediation

Avant de comparer le score, il faut vérifier explicitement qu'il est bien un nombre réel valide

```
if not (0.0 <= score <= 1.0):  
    return "✗ Erreur : score invalide (hors de [0, 1])."
```

Ou normaliser les entrées et refuser toute image contenant NaN ou inf !

```
if not torch.isfinite(image).all():  
    raise ValueError("L'image contient des valeurs invalides (NaN ou Inf).")
```



Fake Image, real impact 1.0

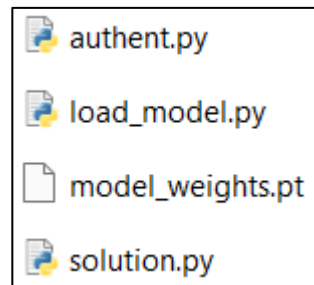
Challenge Inspiré du TRACS 2024

Contexte

Cette fois, tout à bien été sécurisé ! Impossible de désactiver leurs systèmes n'est ce pas ?

Prouvez à votre chef que l'organisation n'est pas hors de notre atteinte. En utilisant les résultats de l'inférence uniquement, hackez leur système. Etrangement il semble que l'authentification réussisse uniquement si le taux d'invalidité de l'image est de `0xDEADBEEF` / `0x2540BE400` avec une précision de 10^{-6} .

Plusieurs fichiers sont disponibles pour ce challenge :



Authent.py vérifie la validité du code QR

Model_weights.pt représente le réseau de neuronne

Load_model.py est un utilitaire pour charger et executer l'IA

Solution.py est un code fourni pour vous aider dans le challenge

Solution

Dans `authent.py`, les informations essentielles à notre objectif sont présentes. Comme d'habitude le réseau de neurones est chargé et passé en mode d'utilisation.

```
def authentication(image_tensor: torch.Tensor, tolerance: float = 1e-6) -> (bool, float): 3 usages
    """
    Renvoie (True, score) si le score d'incertitude renvoyé par le modèle est ≈ 0.373592856
    """
    INVALIDITY_SCORE = 0xDEADBEEF / 0x2540BE400
    image_tensor = image_tensor.to(DEVICE)

    with torch.no_grad():
        out = MODEL(image_tensor)
        score = float(out[0, 0]) #on analyse l'incertitude de l'IA (probabilité p0 que l'image soit invalide)

    if not np.isfinite(score):
        score = 0.0

    valid = abs(score - INVALIDITY_SCORE) <= tolerance
    if valid:
        print FLAG
    else:
        print(f"[-] Access denied (score = {score:.8f})")
```

Le score d'invalidité attendu
On charge l'image sur l'appareil voulu
Calcul du score de l'image

Vérification que le score soit cohérent pour éviter les petits malins

Vérification du score avec une tolérance de 10^{-6}
Obtention du flag

Solution

Commencez par importer les modules et définir l'appareil qui servira le calcul.

```
import torch
import numpy as np
from PIL import Image
from authent import MODEL, authentication

# Configuration
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
MODEL.to(DEVICE)
```

Définissez les constantes pour créer une image

```
# Constantes
TARGET_SCORE = 0xDEADBEEF / 0x2540BE400
IMAGE_SHAPE = (1, 32, 32)
SAVE_PATH = "best_optimized_image.tiff"
TOLERANCE = 1e-6
LEARNING_RATE = 1e-4
MAX_STEPS = 1000
```

Taux d'apprentissage petit pour ne pas créer d'image incohérentes, nombre de pas limités pour éviter une boucle d'optimisation infinie.

Solution

```
def optimize_image(): 1usage
    """Optimise une image aléatoire pour atteindre le score cible."""
    # Création de l'image avec gradient
    image = torch.rand(IMAGE_SHAPE, device=DEVICE, requires_grad=True)
    optimizer = torch.optim.Adam( params: [image], lr=LEARNING_RATE)

    for step in range(1, MAX_STEPS + 1):
        optimizer.zero_grad()

        # Récupère le score du canal 0 (p0 = probabilité que ce n'est PAS un QR code valide)
        score = MODEL(image)[0, 0]

        # Fonction de coût
        loss = (score - TARGET_SCORE) ** 2
        loss.backward()
        optimizer.step()

        # Optionnel : maintenir les pixels dans [0, 1]
        image.data.clamp_( min: 0.0, max: 1.0)

        # Critère de convergence
        if abs(score.item() - TARGET_SCORE) <= TOLERANCE:
            print(f"[+] Convergence atteinte à l'étape {step} (score = {score.item():.8f})")
            break
        else:
            print(f"[-] Pas de convergence après {MAX_STEPS} étapes.")

    return image.detach().cpu()
```

On crée une image vide et on définit l'optimisateur le plus classique en IA

Boucle d'optimisation

On extrait le score d'invalidité

On optimise l'image en calculant la perte

Finalement on vérifie si le score attendu a été atteint, si non, on continue

Renvoyer l'image remis sur le processeur

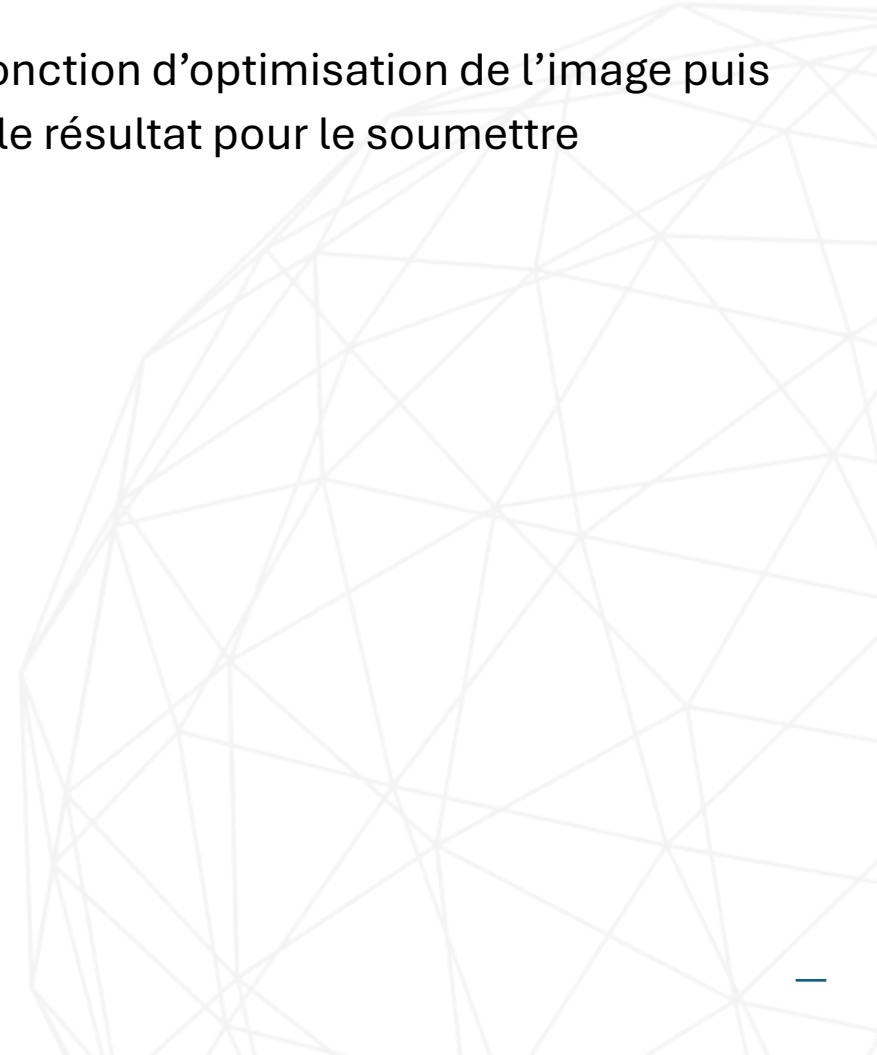
Solution

```
def save_image(image_tensor, path): 1 usage
    """Sauvegarde le tenseur image en TIFF float32 sans conversion en uint8."""
    image_np = image_tensor.squeeze().cpu().numpy().astype(np.float32)
    img = Image.fromarray(image_np, mode='F') # mode F = 32-bit float grayscale
    img.save(path)
    print(f"[+] Image sauvegardée : {path}")

def main(): 1 usage
    image = optimize_image()
    save_image(image, SAVE_PATH)
```

HMP{Giv3_Only_Th3_R3sult} Obtenez le flag !

Exécutez la fonction d'optimisation de l'image puis sauvegardez le résultat pour le soumettre



Remediation

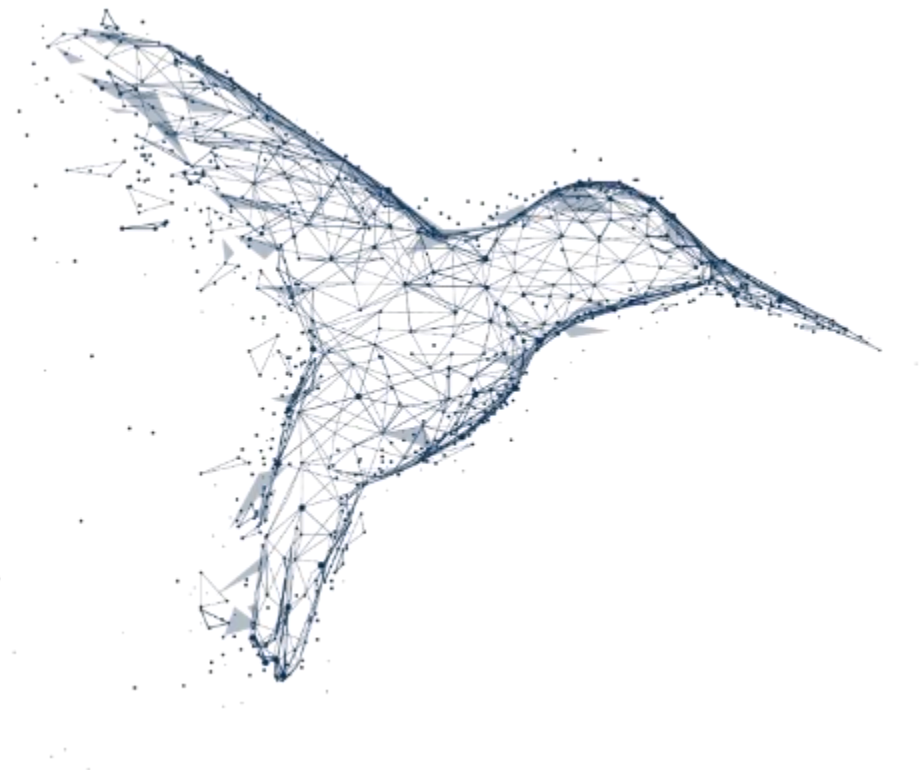
Risque :

Attaque par gradient / optimisation adversariale :

- Un attaquant peut calculer le gradient de la sortie du modèle par rapport à l'entrée, et ajuster progressivement une image pour approcher le score voulu.

Protection :

- Ne laissez pas le score de validation fuiter !
- Limitez le nombre de requêtes
- Ajouter une **signature HMAC** ou un **challenge-réponse cryptographique** au lieu d'une prédiction ML seule.
- Ajouter du bruit ou des défenses adversariales (e.g., détection d'images synthétiques).
- Analyser les entrées pour détecter des patterns générés automatiquement



Fake Image, real impact 2.0

Challenge original de Lucas PETTINATO

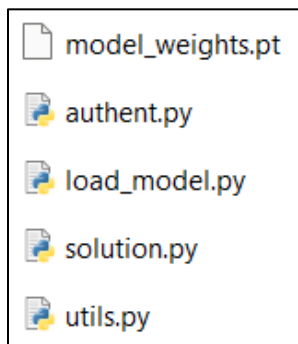
Contexte

L'organisation a décidée de renforcer sa sécurité suite au hack précédent. Passer l'authentification semble très difficile maintenant.

Cependant l'un de nos agents à pu extraire le réseau de neurone !

Le labo vous transmet un code pour vous aider dans votre tâche.

Cinq fichiers sont fournis pour ce challenge :



`model_weights.pt` contient les poids du réseau de neurones.

`authent.py` vérifie la validité du code QR.

`load_model.py` permet de charger et exécuter le modèle IA.

`solution.py` est un squelette de code fourni par le laboratoire à compléter.

`utils.py` est une bibliothèque utilitaire fournie par le laboratoire, contenant les fonctions essentielles au challenge.

Solution

Dans `authent.py`, les informations essentielles à notre objectif sont présentes. Comme d'habitude le réseau de neurones est chargé et passé en mode d'utilisation.

Protection contre les images invalides

```
def authentication(image_tensor: torch.Tensor) -> bool: 2 usages
    if image_tensor.shape != (1, 1, 32, 32):
        return print("❌ Image invalide : mauvaise dimension.") or False

    if not torch.isfinite(image_tensor).all():
        return print("❌ Image invalide : NaN ou Inf détecté.") or False

    if torch.all(image_tensor == image_tensor.view(-1)[0]):
        return print("❌ Image invalide : tous les pixels sont identiques.") or False

    with torch.no_grad():
        score = MODEL(image_tensor)[0, 1].item()
        print(f"Score : {score:.6f}")

        if score >= 0.999:
            print("🚩 Flag : HMP{fake_flag}")
        else:
            print("❌ Score insuffisant.")
```

Score de validité du QRCode de 0 à 100%

Il faut 99,9% de certitude pour valider le challenge

Solution

Dans le fichier Solution.py, des instructions et du code est fourni.
Les imports permettent de guider le challenge :

```
#!/usr/bin/env python3
import os
import torch
import numpy as np
from tifffile import imwrite
from utils import optimize_image
from authent import MODEL, authentication
```

Le format de sauvegarde du QRCode généré sera en **.tiff**

Seule une fonction semble être utile pour nous avec utils.py

Le modèle et le mécanisme d'authentification est chargé depuis authent.py

En allant voir **optimize_image**, Nous avons une description assez complète de la fonction et son utilisation :

```
def optimize_image(model, init_image, adam_steps, lbfgs_steps, 2 usages
    ⚡
    lambda_tv=0.001, layer_idx=0):
    """
    Optimise une image pour maximiser le score de prédiction du modèle.
```

Solution

Rappel : l'objectif est **de trouver une image suffisante, pas nécessairement optimale**.

Une partie du code est déjà fournie ; on identifie clairement une boucle visant à éviter les images piégées dans un minimum local trop peu satisfaisant.

```
RESTARTS = 10
```

```
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
def main(): 1 usage
    # Limiter les threads (utile sur certains clusters ou environnements CPU)
    torch.set_num_threads(1)
    for r in range(RESTARTS):
        |
        # Étape 1 : Créez une image aléatoire initiale
        # -----
        # 🛠 À FAIRE : générer un tenseur aléatoire de forme IMAGE_SHAPE sur le bon DEVICE
        # Exemple : init = ...
```

```
init = torch.rand(IMAGE_SHAPE, device=DEVICE)
```

Solution

```
# Étape 2 : Optimisez cette image avec la fonction fournie
# -----
# 🛠 À FAIRE : appeler optimize_image() avec les bons arguments
# Exemple : img = ...
def optimize_image(model, init_image, adam_steps, lbfgs_steps,
    📢 lambda_tv=0.001, layer_idx=0):
```

Il suffit de taper `help(optimize_image)` pour avoir les instructions de la fonction.

Les arguments de `steps` doivent être **suffisamment grands** pour éviter les minima locaux très mauvais.

```
img = optimize_image(MODEL, init, adam_steps=300, lbfgs_steps=100)
```


Vous pouvez consulter `authent.py` pour reproduire son comportement voir appeler directement la fonction d'authentification.

```
# Étape 3 : Calculez le score pour la classe cible
# -----
# 🛠 À FAIRE : extraire le score de sortie de la classe d'intérêt (ex: index 1)
# Exemple : score = ...
```

```
score = MODEL(img)[0, 1].item()
print(f"Score obtenu : {score:.6f}")
```

Solution

```
# Étape 4 : Sauvegarde de l'image optimisée  
# -----  
# 🛠 À FAIRE : convertir l'image en array numpy 2D et sauvegarder en TIFF
```



```
arr = best_img.cpu().numpy().squeeze(0).astype(np.float32)  
imwrite(SAVE_PATH, arr)  
print(f"\n✅ Image enregistrée : {SAVE_PATH}")
```

Transmettez l'image au CPU puis transformez-la en tableau numpy (format classique float32) puis sauvegardez

Solution

Enfinement voici une routine complète avec une sélection automatique de la meilleure image trouvée :

```
def main(): 1usage
# Limiter les threads (utile sur certains clusters)
torch.set_num_threads(1)

# Initialisation
best_score = -float('inf')
best_img = None

print("[*] Génération d'image optimisée...")

# Boucle principale de tentative d'optimisation
for r in range(RESTARTS):
    print(f"\n[ Tentative {r+1}/{RESTARTS} ]")
    init = torch.rand(IMAGE_SHAPE, device=DEVICE)
    img = optimize_image(MODEL, init, adam_steps=300, lbfgs_steps=100)

    score = MODEL(img)[0, 1].item()
    print(f"Score obtenu : {score:.6f}")

    # Mise à jour de la meilleure image
    if score > best_score:
        best_score = score
        best_img = img
```

```
# Vérification que l'on a bien une image optimisée
if best_img is None:
    print("❌ Échec : aucune image valide générée.")
    return

arr = best_img.cpu().numpy().squeeze(0).astype(np.float32)
imwrite(SAVE_PATH, arr)
print(f"\n✅ Image enregistrée : {SAVE_PATH}")

# Vérification avec le système d'authentification
reloaded = imread(SAVE_PATH)
if reloaded.ndim == 3:
    reloaded = reloaded.squeeze(0)

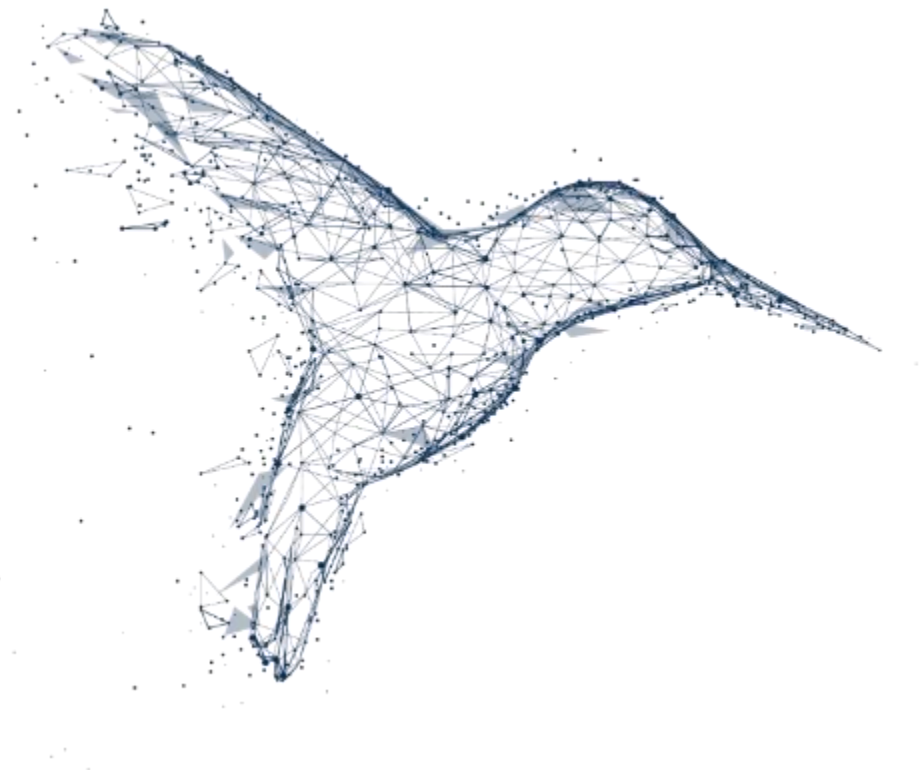
tensor = torch.from_numpy(reloaded).unsqueeze(0).unsqueeze(0).float().to(DEVICE) # ✅ ICI
authentication(tensor)
```

HMP{Prot3ct_Your_N3ural_N3twork}

Remediation

- **Limiter l'exposition du modèle :**
 - API uniquement (Ne distribue jamais les poids ou l'architecture du modèle)
 - Rate limiting & monitoring (Applique des limites sur le nombre de requêtes.)
 - Surveille les patterns d'appel suspects (ex. : balayage intensif de l'espace d'entrée)
- **Chiffrement du modèle au repos**
- **Obfuscation du modèle :**
 - couches leurres
 - Bruit aléatoire
 - fusion de couches
- **Compilation et compression :**
 - Quantization (dégrade la lisibilité directe)
 - Pruning (supprime les connexion inutiles)
 - Compiled Models (rend plus difficile l'analyse du modèle)





Partial Softmax Leak, secret face

Challenge original de Lucas PETTINATO

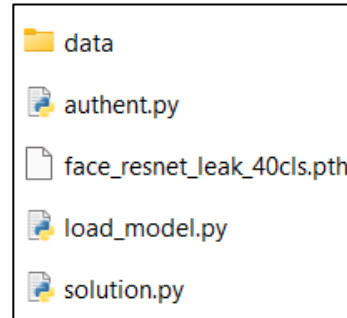
Contexte

Todo : rendre la solution plus accessible pour un ctf

Nous avons pu capturer une IA qui permet de reconnaître le visage du chef, mais impossible de le trouver. Pouvez vous trouver le coupable ?

L'objectif ici est d'analyser une banque d'images de suspects. Nous avons en sortie du modèle un couche softmax partielle.

Plusieurs fichiers sont fournis pour ce challenge :



Data est un dossier avec les visages des suspects

Face_resnet_leak_40cls.pth contient les poids du réseau de neurones.

authent.py vérifie la validité du code QR.

load_model.py permet de charger et exécuter le modèle IA.

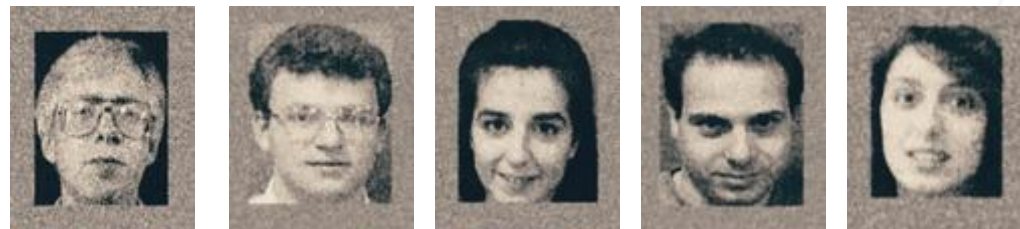
solution.py est un squelette de code fourni par le laboratoire à compléter. (Toujours complet ici, supprimer des parties pour le rendre intéressant pour le challenge)

Solution

Dans `authent.py`, ce n'est pas directement le résultat de l'inférence qui est renvoyé, mais la sortie du softmax, c'est-à-dire le score de reconnaissance du visage comparé aux autres visages rencontrés pendant l'entraînement. **Ici, rendre le fichier inaccessible directement.**

Il suffit d'appeler `authent.py` et de lui soumettre une image. On constate qu'il n'y a que 35 scores pour 40 images, donc 5 sont cachés.

```
result = authentication("data/0.PNG")  
print(len(result)) #donne 35
```



Solution

Il va falloir faire une analyse statistique car ce genre de modèle sont facilement surentrainés pour reconnaître 1 seul visage comme le bon.

Avec seulement 5 suspects l'objectif est de trouver celui qui active le plus l'IA, qui semble le plus suspect.

```
# Score combiné
def combined_score(p: np.ndarray) -> float: 1usage
    """
    Combine un score de confiance et d'incertitude :
    score = peakiness / entropie
    """
    e = entropy(p)
    k = peakiness(p)
    return k / (e + 1e-6)
```

Trois fonctions sont fournies, une nous intéresse vraiment : celle qui calcule le score combiné !

Solution

```
def solve_via_combined(image_paths: list[str]) -> tuple[int, list[float]]:
    """
    Pour chaque image, calcule le score combiné sur le leak.
    Renvoie l'indice de l'image avec score maximal.
    """
    scores = []
    for path in image_paths:
        p = authentication(path)
        scores.append(combined_score(p))
    chef_index = int(np.argmax(scores))
    return chef_index, scores
```



Solution

Finalement, après analyse statistique, nous pouvons constater que c'est le visage ID 0 qui est le chef de l'organisation.

