WriteUp - CryptoBro en détresse

Auteur: Morad Halmi - École2600

Date: 21 avril 2025

Catégorie: Side-Channel Analysis / Hardware Security

Description du challenge

Le challenge "CryptoBro en détresse" propose une situation où un utilisateur a besoin de retrouver le code PIN de son portefeuille de cryptomonnaies. Pour ce faire, il a réalisé des acquisitions avec un oscilloscope pour chaque PIN possible (de 0000 à 9999), mais coupe l'alimentation après quelques microsecondes pour éviter de déclencher un mécanisme de sécurité.

Le challenge nous fournit une archive contenant 10 000 fichiers de traces nommés `trace_XXXX.npy`, où XXXX correspond au PIN utilisé pour générer la trace. Notre mission est d'analyser ces traces pour identifier le PIN correct.

Comprendre le problème

Ce challenge est une simulation d'attaque par canal auxiliaire (side-channel attack) classique. Ce type d'attaque exploite les caractéristiques physiques d'un système de sécurité, comme:

- La consommation électrique
- Les émissions électromagnétiques
- Le temps d'exécution

Dans notre cas, les traces d'oscilloscope capturent probablement des différences subtiles dans le comportement électrique du système lors de la vérification du PIN.

Approche

Mon approche consiste à analyser les traces pour identifier des différences caractéristiques qui pourraient révéler le PIN correct. L'hypothèse principale est que la vérification du PIN se fait probablement chiffre par chiffre, et que le système montre un comportement différent lorsqu'un chiffre correct est vérifié.

Étape 1: Chargement et préparation des données

J'ai d'abord créé un script Python pour charger les traces et les analyser:

python

```python
import numpy as np
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
from collections import defaultdict

def load_traces(directory):
    """
    Charge toutes les traces depuis le répertoire spécifié
    """
    traces = {}
    print("Chargement des traces...")

    # Listez tous les fichiers de trace
    files = [f for f in os.listdir(directory) if f.startswith('trace_') and f.endswi

    for filename in tqdm(files):
        # Extrait le PIN depuis le nom de fichier (trace_XXXX.npy)
        pin = filename.split('_')[1].split('.')[0]

        # Charge la trace
        trace_path = os.path.join(directory, filename)
        trace = np.load(trace_path)

        traces[pin] = trace

    print(f"Chargé {len(traces)} traces.")
    return traces
```

Étape 2: Analyse chiffre par chiffre

L'idée principale est d'analyser chaque position du PIN indépendamment. Pour chaque position:

1. Regrouper les PINs par chiffre à cette position
2. Calculer la moyenne des traces pour chaque groupe
3. Analyser les différences entre ces moyennes pour identifier le chiffre qui se comporte différemment

python

```python
def digit_by_digit_analysis(traces):
    """
    Analyse rigoureuse chiffre par chiffre des traces
    """
    pins = list(traces.keys())
    pin_length = len(pins[0])
    discovered_pin = ""

    # Pour chaque position dans le PIN
    for pos in range(pin_length):
        print(f"\n===== Analyse de la position {pos+1} du PIN =====")

        # Filtrer les PINs qui ont les mêmes chiffres que ce qu'on a découvert jusqu
        if pos > 0:
            filtered_pins = [p for p in pins if p.startswith(discovered_pin)]
            print(f"Filtrage des PINs: {len(filtered_pins)} PINs ont le préfixe '{di
        else:
            filtered_pins = pins

        # Pour cette position, regrouper les PINs par chiffre
        digit_groups = defaultdict(list)
        for pin in filtered_pins:
            digit = pin[pos]
            digit_groups[digit].append(pin)

        # Calculer la moyenne des traces pour chaque groupe
        digit_means = {}
        for digit, group_pins in digit_groups.items():
            group_traces = [traces[pin] for pin in group_pins]
            digit_means[digit] = np.mean(group_traces, axis=0)

        # Calculer la variance globale entre les moyennes des groupes
        all_means = np.array(list(digit_means.values()))
        variance_between_means = np.var(all_means, axis=0)

        # Trouver les points de haute variance
        high_variance_points = np.argsort(variance_between_means)[-50:]

        # Analyser la séparation entre les groupes aux points de haute variance
        separation_scores = {}
        for digit1, mean1 in digit_means.items():
            total_distance = 0
            count = 0

            for digit2, mean2 in digit_means.items():
                if digit1 != digit2:
```

```
                    # Calculer la distance euclidienne aux points de haute variance
                    distance = np.linalg.norm(mean1[high_variance_points] - mean2[hi
                    total_distance += distance
                    count += 1

            if count > 0:
                separation_scores[digit1] = total_distance / count

    # Trier les chiffres par score de séparation
    sorted_scores = sorted(separation_scores.items(), key=lambda x: x[1], revers

    # Sélectionner le chiffre avec le meilleur score
    best_digit = sorted_scores[0][0]
    discovered_pin += best_digit

return discovered_pin
```

Étape 3: Vérification approfondie

Pour confirmer nos résultats, j'ai implémenté une vérification supplémentaire qui compare le PIN candidat avec d'autres PINs qui diffèrent à chaque position:

python

```python
def in_depth_verification(traces, pin_candidate):
    """
    Vérification approfondie d'un PIN candidat
    """
    pins = list(traces.keys())
    pin_length = len(pin_candidate)

    # Pour chaque position, comparer ce PIN avec d'autres qui diffèrent à cette posi
    for pos in range(pin_length):
        # Trouver des PINs qui diffèrent uniquement à cette position
        similar_pins = []

        for pin in pins:
            if pin != pin_candidate:
                # Vérifier si ce PIN diffère uniquement à la position courante
                differs_only_at_pos = True
                for i in range(pin_length):
                    if i != pos and pin[i] != pin_candidate[i]:
                        differs_only_at_pos = False
                        break

                if differs_only_at_pos:
                    similar_pins.append(pin)

        # Calculer la différence moyenne entre le PIN candidat et ces PINs similaire
        diffs = []
        for pin in similar_pins:
            diff = np.abs(traces[pin_candidate] - traces[pin])
            diffs.append(diff)

        if diffs:
            mean_diff = np.mean(diffs, axis=0)
            # Trouver les points où la différence est la plus marquée
            top_diff_points = np.argsort(mean_diff)[-20:]
```

Résultats et analyse

L'exécution de mon script a révélé des informations extrêmement intéressantes:

Position 1 (premier chiffre)

- Le chiffre 9 avait un score de séparation de 1.704610, nettement supérieur aux autres chiffres
- Le deuxième meilleur score était de seulement 0.213138 (chiffre 2)
- Cette différence marquée indique clairement que 9 est le premier chiffre

Position 2 (deuxième chiffre)

- Pour les PINs commençant par 9, le chiffre 4 avait un score de 1.630508
- Les autres chiffres étaient tous sous 0.22
- Conclusion: le deuxième chiffre est 4

Position 3 (troisième chiffre)

- Pour les PINs commençant par 94, le chiffre 6 avait un score de 1.518194
- Les autres scores étaient beaucoup plus bas
- Conclusion: le troisième chiffre est 6

Position 4 (quatrième chiffre)

- Pour les PINs commençant par 946, le chiffre 6 avait un score de 1.335019
- Les autres chiffres avaient des scores significativement plus bas
- Conclusion: le dernier chiffre est 6

Le PIN identifié est donc 9466, ce qui donne le flag FCSC{9466}.

Vérification

Pour confirmer ce résultat, j'ai comparé le PIN 9466 avec d'autres PINs qui diffèrent à chaque position (comme 9467, 9468, etc.). L'analyse des différences a confirmé que 9466 présente des caractéristiques uniques à chaque position.

Leçons apprises

Cette attaque par canal auxiliaire démontre plusieurs points importants:

1. **Faiblesses des implémentations matérielles**: Même un système qui coupe l'alimentation rapidement peut fuir des informations via les canaux auxiliaires.
2. **Importance de l'analyse statistique**: L'utilisation de techniques statistiques avancées peut révéler des différences subtiles dans les traces.
3. **Analyse séquentielle**: Les systèmes qui vérifient un PIN chiffre par chiffre sont vulnérables à des attaques qui exploitent cette séquentialité.
4. **Sécurité matérielle**: Le challenge illustre l'importance de la sécurité au niveau matériel, pas seulement logiciel.

Conclusion

Ce challenge illustre parfaitement comment des fuites d'information par canaux auxiliaires peuvent compromettre la sécurité d'un système cryptographique, même avec des mesures de protection comme

la coupure rapide de l'alimentation.

La méthodologie utilisée ici pourrait s'appliquer à d'autres systèmes embarqués sécurisés comme les cartes à puce, les tokens d'authentification ou les portefeuilles matériels de cryptomonnaies.

Pour améliorer la sécurité contre ce type d'attaques, les concepteurs de systèmes devraient:

- Implémenter des contre-mesures contre les attaques par canaux auxiliaires (masquage, exécution constante)
- Éviter les vérifications séquentielles des secrets
- Ajouter du bruit aléatoire dans la consommation électrique

Le flag final est: **FCSC{9466}**