💻 Hansung Code Camp Problems

Problem 1 - 디지털 포렌식 문제

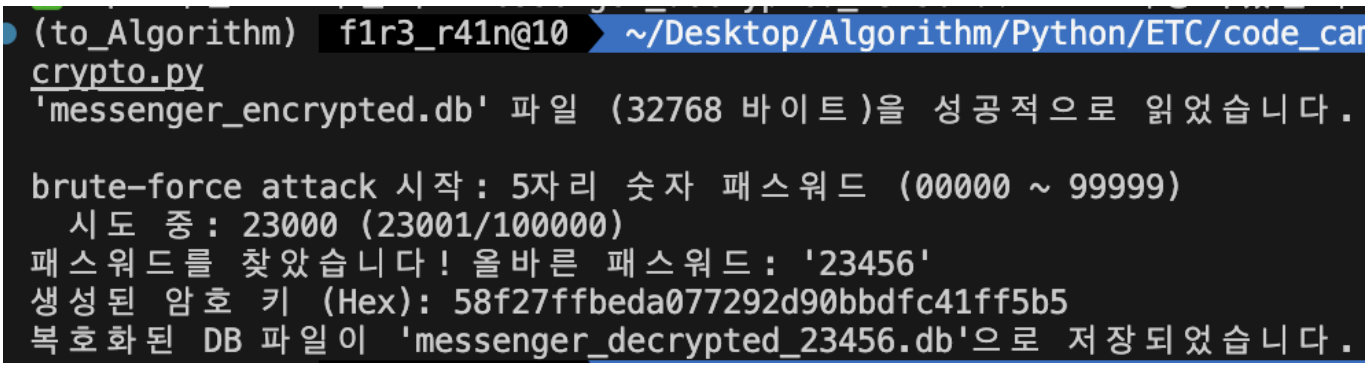
문제 정보

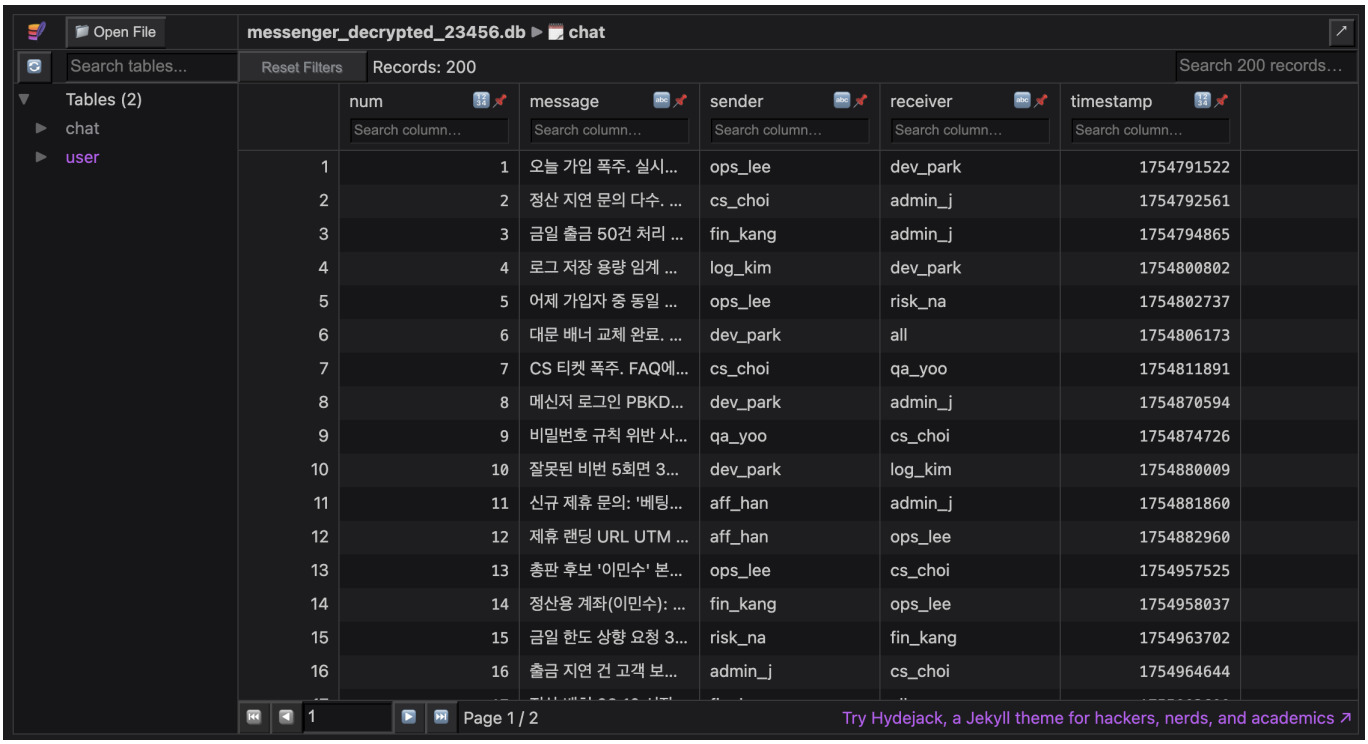
- 메신저 실행 시 **사용자가 입력하는 숫자(0~9)로 이루어진 5자리 패스워드**
- **PBKDF2-HMAC-SHA256 방식을 통해 16바이트의 암호 키 생성**
  - 키 생성 알고리즘: PBKDF2-HMAC-SHA256(Password, Salt=0x00*16, IterationCount=10,000, DerivedKeyLength=16)
- 평문을 암호 키와 **16바이트 단위의 XOR 연산**을 통해 암/복호화 설계
- 올바른 암호 키로 첫 16바이트 복호화 시 다음의 평문 정보 식별
  - `53 51 4C 69 74 65 20 66 6F 72 6D 61 74 20 33 00 : SQLite format 3.`

문제 풀이 포인트

- 0~9 숫자로 이루어진 5자리 패스워드이기 때문에 Brute-Force Attack이 가능함
- 파이썬에서의 PBKDF2-HMAC-SHA256 방식의 메서드는 `hashlib` 모듈의 `pbkdf2_hmac` 메서드 사용
- 생성 알고리즘의 정보들은 상수로 설정
- 생성된 암호 키로 평문을 XOR 연산하는 메서드 따로 구현
- 파일 전체가 아닌 앞 부분의 시그니처 복호화 후 sqlite3 파일의 시그니처와 일치하는 지 확인
- 일치하면 비밀번호와 복호화된 파일 생성

결과





Open File | messenger_decrypted_23456.db ▶ chat

Search tables... | Reset Filters | Records: 200 | Search 200 records...

Tables (2)
- chat
- user

	num	message	sender	receiver	timestamp
	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	오늘 가입 폭주. 실시...	ops_lee	dev_park	1754791522
2	2	정산 지연 문의 다수. ...	cs_choi	admin_j	1754792561
3	3	금일 출금 50건 처리 ...	fin_kang	admin_j	1754794865
4	4	로그 저장 용량 임계 ...	log_kim	dev_park	1754800802
5	5	어제 가입자 중 동일 ...	ops_lee	risk_na	1754802737
6	6	대문 배너 교체 완료. ...	dev_park	all	1754806173
7	7	CS 티켓 폭주. FAQ에...	cs_choi	qa_yoo	1754811891
8	8	메신저 로그인 PBKD...	dev_park	admin_j	1754870594
9	9	비밀번호 규칙 위반 사...	qa_yoo	cs_choi	1754874726
10	10	잘못된 비번 5회면 3...	dev_park	log_kim	1754880009
11	11	신규 제휴 문의: '베팅...	aff_han	admin_j	1754881860
12	12	제휴 랜딩 URL UTM ...	aff_han	ops_lee	1754882960
13	13	총판 후보 '이민수' 본...	ops_lee	cs_choi	1754957525
14	14	정산용 계좌(이민수): ...	fin_kang	ops_lee	1754958037
15	15	금일 한도 상향 요청 3...	risk_na	fin_kang	1754963702
16	16	출금 지연 건 고객 보...	admin_j	cs_choi	1754964644

1 | Page 1 / 2 | Try Hydejack, a Jekyll theme for hackers, nerds, and academics ↗

Problem 2 - 디지털 포렌식 문제

- apk 파일 분석 후 암호화된 사진을 복호화하여 증거 확보
- 사용자 비밀번호 입력 후, 해당 암호 기반으로 키를 파생하여 파일 AES 암호화
    - 별도 난독화 존재 X
- AES 키 파생 알고리즘: PBDKF2-HMAC-SHA1
    - 파생 키로 AES-128-CTR 알고리즘으로 암호화 수행

문제 풀이 포인트

```java
/* loaded from: classes3.dex */
public class Crypto {
    private static final int KEY_BITS = 128;
    private static final int PBKDF2_ITER = 1000;
    private static final String TRANSFORM = "AES/CTR/NoPadding";
    private static final String SALT_HEX = "c8570ac98cc615aa6a6b97b3f20f1b41";
    private static final byte[] SALT = hexToBytes(SALT_HEX);

    public static byte[] deriveKeyFromPassword(char[] password) throws Exception {
        PBEKeySpec spec = new PBEKeySpec(password, SALT, 1000, 128);
        SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        return skf.generateSecret(spec).getEncoded();
    }

    public static byte[] encrypt(byte[] plain, byte[] keyBytes) throws Exception {
        SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
        IvParameterSpec iv = new IvParameterSpec(keyBytes);
        Cipher cipher = Cipher.getInstance(TRANSFORM);
        cipher.init(1, key, iv);
        return cipher.doFinal(plain);
    }

    public static byte[] decrypt(byte[] cipherData, byte[] keyBytes) throws Exception {
        SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
        IvParameterSpec iv = new IvParameterSpec(keyBytes);
        Cipher cipher = Cipher.getInstance(TRANSFORM);
        cipher.init(2, key, iv);
        return cipher.doFinal(cipherData);
    }

    private static byte[] hexToBytes(String hex) {
        String clean = hex.toLowerCase().replaceAll("[^0-9a-f]", "");
        if (clean.length() % 2 != 0) {
            throw new IllegalArgumentException("Invalid hex");
        }
        byte[] out = new byte[clean.length() / 2];
        for (int i = 0; i < out.length; i++) {
            int hi = Character.digit(clean.charAt(i * 2), 16);
            int lo = Character.digit(clean.charAt((i * 2) + 1), 16);
            out[i] = (byte) ((hi << 4) | lo);
        }
        return out;
    }
}
```

- iter: 1,000
- salt: `c8570ac98cc615aa6a6b97b3f20f1b4`
- png 파일의 시그니처는 `89504E47 0D0A1A0A`
- 고정된 salt 값과 인코딩된 파일의 IV가 계속 반복되는 키 재사용성이라는 취약점 때문에 XOR 연산으로 사용자의 비밀번호를 알아낼 수 있음

Problem 3 - 머신러닝 기반 암호화 트래픽 분석

- 암호화 트래픽을 인공지능 모델의 입력으로 사용하기 적절하도록 전처리 수행

    - 전처리 방법론에 따라 제시된 pcap 파일을 전처리(flow 추출, 특징 구성, 정규화)를 수행하여 머신러닝 모델에 학습시킬 수 있도록 데이터셋 구성
- 전처리 방법론
    - 네트워크 트래픽 데이터는 일반적으로 .pcap 파일 포맷으로 저장되어 있으며, 여러 flow가 혼재되어 있음
    - Flow-level 특징을 추출하기 위해 트래픽 데이터를 Flow 별로 분할
        - 5-Tuple(Source IP, Destination IP, Source Port, Destination Port, Protocol)을 기준으로 분할
        - 모델의 입력으로 사용하기 위해서는 데이터셋을 고정된 길이, 동일한 형식으로 정규화 필요
            - Standard Scaler
            - Min-Max Scaler
- 머신러닝 모델의 입력으로 사용하기 위해 다음과 같은 단계로 암호화 트래픽 데이터 전처리를 수행했습니다.
- pcap 파일 로드: 제공된 '/content/benign.pcap' 및 '/content/malware.pcap' 파일을 Scapy 라이브러리를 사용하여 로드했습니다. 로드된 총 패킷 수는 **{len(packets)}**개 입니다. (정상: {len(benign_packets)}개, 악성: {len(malware_packets)}개)
- 흐름 추출 및 특징 구성: 로드된 패킷에서 데이터 흐름(flow)을 식별하고 각 흐름에 대한 특징을 추출했습니다. 총 **{len(flows)}**개의 흐름이 식별되었으며, 각 흐름에 대해 흐름 특징과 레이블(flow_label)을 할당했습니다.
- 데이터 구조화: 추출된 흐름 특징들은 pandas DataFrame (flows_df) 형태로 구조화되었습니다. 이 DataFrame은 총 **{flows_df.shape[0]}**개의 행(흐름)과 **{flows_df.shape[1]}**개의 열(특징 및 레이블)으로 구성됩니다.
- 데이터 정규화: 숫자형 특징들에 대해 MinMaxScaler를 사용하여 정규화를 수행했습니다.
- 데이터셋 분할: 정규화된 데이터셋은 학습 세트와 테스트 세트로 분할되었습니다. 학습 세트는 {X_train.shape}, 테스트 세트는 **{X_test.shape}**의 형태를 가집니다. 레이블(y_train, y_test)도 각각 {y_train.shape}, {y_test.shape} 형태입니다.

Problem 4 - 머신러닝 기반 암호화 트래픽 분석

- 전처리된 암호화 트래픽 데이터를 머신러닝 모델을 사용하여 악성 트래픽인지 또는 정상 트래픽인지를 분류
    - Random Forest, Logistic Regression 모델을 사용하여 전처리한 데이터셋을 학습시켜 성능을 확인
- 전처리된 데이터셋을 사용하여 Random Forest 및 Logistic Regression 모델을 학습시키고 악성 트래픽 분류 성능을 평가했습니다.
- Random Forest 모델: 학습 결과, 테스트 데이터에 대한 성능은 **정확도 1.0000과 F1 점수 1.0000**을 기록했습니다.
- Logistic Regression 모델: 학습 결과, 테스트 데이터에 대한 성능은 **정확도:1.0000과 F1 점수 1.0000**을 기록했습니다.

Problem 5 - 암호화

- AES로 암호화된 데이터를 복구하라
- 적절한 암호화 방식을 선택하라

```
# cipher = AES.new(key, AES.MODE_CFB, iv=iv)
# cipher = AES.new(key, AES.MODE_OFB, iv=iv)
# cipher = AES.new(key, AES.MODE_CTR, nonce=nonce)  # ← 여기!
```

- 패딩이 필요 없고, 스트림 방식(키스트림과 XOR, "counter increases")을 사용한다는 점에서 CTR (Counter) 모드가 정답입니다.

- 암호화 키값을 유추하시오.
    - 최종 키 문자열 codingcryptogra#

Problem 6 - 암호화

- Low Public Exponent Attack을 RSA에 적용하여 해킹하시오.
    - 적절한 e 값 대입하여 복호화

```python
# 구글 코랩에서 gmpy2 설치
!pip install gmpy2

# 설치 후 import
import gmpy2

# RSA Low Public Exponent Attack
# 암호문
c = 342516537566644178912600085687494155978384636108462281062550912835303050636279390674990

print("구글 코랩에서 RSA Low Public Exponent Attack")
print("=" * 60)
print(f"암호문 c = {c}")
print(f"암호문 자릿수: {len(str(c))}")
print("\n작은 e값들을 시도합니다...")

# e값을 1부터 100까지 시도
found = False
for e in range(1, 101):
    # e제곱근 구하기
    m, exact = gmpy2.iroot(c, e)

    if exact:
        print(f"\n🎯 e = {e}에서 정확한 제곱근 발견!")
        print(f"m = {m}")

print("\n" + "=" * 60)
print("Low Public Exponent Attack 완료!")
print("공격 원리: m^e < n일 때 c = m^e이므로 c의 e제곱근 = m")
```

- 발견된 키: e = 17
- 복호화된 메시지: "camp2025"