Assignment 3

Information Security in 2025 Fall

Oh Jiwoo
Student ID: 2022428418

Kyungpook National University

November 12, 2025

Contents

Content summary **2**

1 Introduction **2**

2 Background Theory **2**
2.1 Encryption and Decryption of CBC 2
2.2 PKCS#7 Padding Method 3
2.3 Theory of Padding Oracle Attack 3

3 Implementation **3**
3.1 Oracle Wrapper 3
3.2 util 3
3.3 Core Algorithm 4
3.4 main code 5

4 Results **6**

5 Conclusion **7**
5.1 목표에 대한 달성 여부 7
5.2 발생 가능한 문제 7
5.3 시사점 7

Content summary

- **Introduction**
  연구의 목적과 과제에 대한 설명
- **Background Theory**
  CBC 동작 원리, PKCS#7 패딩 규칙, 패딩 오라클 공격에 대한 이론 서술
- **Implementation**
  주요 알고리즘 흐름, 오라클 래퍼와 유틸 함수, 핵심 복구 로직, 실행부 코드 설명
  - *Oracle Wrapper* — 제공된 오라클 API를 어떻게 래핑했는지.
  - *Utility* — 16진법 변환, 패딩 제거 등 보조 함수.
  - *Core Algorithm* — 복호화 알고리즘
  - *Main* — main 코드
- **Results**
  실제 복원된 평문을 스크린샷으로 제시
- **Conclusion**
  실험의 달성도, 발생 가능한 문제점, 보안적 시사점에 대해 서술

1 Introduction

본 과제의 목적은 *패딩 오라클 공격 (Padding Oracle Attack)*을 직접 구현하고, 이를 통해 제공받은 암호문으로부터 평문을 복원하는 것에 있다. CBC(Cipher Block Chaining) 모드에서 패딩 오류의 정보를 활용해 암호문의 내용을 전혀 모르는 상태에서도 평문으로 복호화할 수 있는 취약점을 과제를 통해 알 수 있다. 이를 수행하기 위해, 패딩 오라클의 인터페이스를 Java로 호출했고, PKCS#7 패딩 방식과 CBC 모드 동작 원리를 고려해 코드를 설계했다. 앞으로 보고서에서는 패딩 오라클 공격이 무엇인지에 대한 개념과 배경이 되는 이론에 대해 설명한 다음, 구현 프로그램에 대해 상세히 기술할 것이다. 또한, 과제 수행을 통해서 얻은 보안적인 시사점에 대해서도 제시해볼 것이다.

2 Background Theory

2.1 Encryption and Decryption of CBC

CBC 모드는 블록 암호 방식 중의 하나로, 각 평문 블록을 이전 암호문의 블록과 XOR 연산하여 암호화하는 방식이다. CBC 모드는 위와 같이 이전 단계의 암호문이 현재 블록 평문을 알아내는 데 사용되기 때문에 특정 암호문 블록을 조작하게 되면 다음 블록의 복호화의 결과 일부를 알 수 있다. 특히, 이 특징이 본 과제의 주제인 패딩 오라클 공격에 사용된다. 또한, CBC 모드는 암호화를 처음 시작할 때마다 IV를 생성해 사용한다. 첫 블록 이후의 블록들은 이전 암호문 블록과 XOR 연산을 통해 계산된다. 현재 과제에서 사용할 블록과 키는 8byte 크기다. 그래서 한 블록에 8byte를 담을 수 있기 때문에 평문의 길이가 8byte보다 작으면 패딩으로 크기를 맞춰야 한다.

2.2 PKCS#7 Padding Method

평문 길이가 블록 크기로 정확히 나누어떨어지지 않을 때 남는 공간을 패딩으로 채우는데, 이때 PKCS#7은 널리 사용되고 있는 방식 중의 하나이다. 예를 들어, 채워야 하는 공간이 2 바이트라고 하면 0x02, 0x02를 각각 추가한다. 만약 정확히 8byte라고 할 때는 8byte의 새 블록을 추가하고, 나머지는 전부 0x08로 패딩한다. 복호화를 할 때에는 마지막 바이트 값을 확인해서 패딩이 오류가 있는지 아닌지를 판단한다.

2.3 Theory of Padding Oracle Attack

패딩 오라클 공격은 위의 CBC 모드와 PKCS#7 패딩이 같이 사용될 때, 패딩 오류 질의에 응답하는 오라클을 사용해 평문을 알아내는 공격 방식이다.

3 Implementation

해당 프로그램은 C0와 C1을 8바이트 단위의 바이트 배열로 변환한 뒤, pad_oracle을 이용해 가능한 C0의 값을 탐색한다. 이후 C1의 각 바이트를 유추하여 C1과 C0을 XOR 연산해 최종 평문을 복구하고, 패딩을 제거하여 ASCII로 출력한다. 구현의 큰 기능으로는 Oracle 호출을 위한 Wrapper와 16진수를 처리하기 위한 util 함수, 패딩 오라클 공격의 주요 알고리즘과 main code(실행부)로 나누어진다.

3.1 Oracle Wrapper

```
static class Oracle {
    private final pad_oracle oracleInstance;

    public Oracle() {
        this.oracleInstance = new pad_oracle();
    }

    public boolean check(String c0hex, String c1hex) {
        return oracleInstance.doOracle(c0hex, c1hex);
    }
}
```

제공받은 pad_oracle.jar의 pad_oracle을 감싸는 wrapper의 역할을 한다. 오라클에 질의하여 패딩의 참/거짓 값을 반환하게 한다. 맞으면 1, 틀리면 0을 반환하는 방식으로 동작한다.

3.2 util

```
static byte[] hexToBytes(String s) {
    String n = s.trim();
    if (n.startsWith("0x")) n = n.substring(2);
    if ((n.length() % 2) == 1) n = "0" + n;
    byte[] out = new byte[n.length() / 2];
    for (int i = 0; i < out.length; i++) {
        out[i] = (byte) Integer.parseInt(n.substring(2 * i, 2 * i + 2), 16);
    }
    return out;
}
```

인자 두 개를 받아서 0x를 제거한 다음, 홀수면 앞에 0을 붙인다. 문자열을 parseInt로 16진수 값을 얻어내고, 바이트 배열을 만들어낸다.

```
static String bytesToHex(byte[] b) {
    StringBuilder sb = new StringBuilder();
    for (byte x : b) sb.append(String.format("%02x", x & 0xff));
    return sb.toString();
}

static byte[] removePadding(byte[] block) throws Exception {
    if (block.length == 0) return block;
    int pad = block[block.length - 1] & 0xff;
    if (pad <= 0 || pad > block.length) throw new Exception("Invalid padding");
    for (int i = block.length - pad; i < block.length; i++) {
        if ((block[i] & 0xff) != pad) throw new Exception("Invalid padding");
    }
    return Arrays.copyOf(block, block.length - pad);
}
```

util 부분 코드는 16진수 문자열을 byte 배열로 변환하고, 발견 평문 byte를 누적하고, 복구 완료 후 패딩을 제거하는 기능을 한다.

3.3 Core Algorithm

```
static byte[] recoverBlock(byte[] C0, byte[] C1, Oracle oracle) {
    final int B = C0.length;

    byte[] recovered = new byte[B];
    Arrays.fill(recovered, (byte) 0);

    java.util.function.BiFunction<Integer, byte[], List<Integer>>
    list = (idx, cur) -> {
        int i = idx;
        int p = B - i;
        List<Integer> ok = new ArrayList<>();
        for (int g = 0; g < 256; g++) {
            byte[] C0p = Arrays.copyOf(C0, B);
            for (int j = B - 1; j > i; j--) {
                C0p[j] = (byte) ((C0[j] ^ cur[j]) ^ p);
            }
            C0p[i] = (byte) g;
            boolean res = oracle.check("0x" + bytesToHex(C0p),
            "0x" + bytesToHex(C1));
            if (res) ok.add(g);
        }
        return ok;
    };

    class DFS {
        boolean dfs(int i, byte[] cur) {
```

```
            if (i < 0) {
                System.arraycopy(cur, 0, recovered, 0, B);
                return true;
            }
            List<Integer> find = list.apply(i, cur);
            if (find.isEmpty()) return false;
            int p = B - i;
            for (int n : find) {
                byte[] next = Arrays.copyOf(cur, B);
                int m = (n ^ p) & 0xff;
                next[i] = (byte) (m ^ (C0[i] & 0xff));
                if (dfs(i - 1, next)) return true;
            }
            return false;
        }
    }

    boolean ok = new DFS().dfs(B - 1, new byte[B]);
    return recovered;
}
```

위에서 CBC 블록에 대한 평문 복원이 이루어진다. 1부터 8byte로 크기를 늘려가면서 반복해서 찾는다. 평문 마지막 byte 부터 차례대로 구한다. 여기서 최대 256번의 서치가 byte마다 이뤄질 수 있다. 각 단계마다 패딩 길이를 정해서 IV의 뒤쪽 바이트를 조작해서 패딩 패턴을 만들어낸다. 처음 단계에서는 마지막 byte만 올바른 것으로 바꿔서 오라클에 쿼리를 던진다. 오라클이 1을 반화하면 그때 값이 C1이 마지막 byte와 0x01의 XOR 연산을 수행한 것이라는 점을 활용하여, 평문의 마지막 byte를 계산한다. 패딩의 길이가 2가 되는 두 번째 단계에서는 마지막 두 바이트를 0x02에 맞게 조정한다. 그리고 그 앞의 바이트를 0부터 255까지 바꿔가면서 계속해서 질의를 한다. 이 단계를 8단계를 거치면 완전한 평문의 복호화가 가능하다.

3.4 main code

```
public static void main(String[] args) throws Exception {

    byte[] C0 = hexToBytes(args[0]);
    byte[] C1 = hexToBytes(args[1]);

    Oracle oracle = new Oracle();
    byte[] P1 = recoverBlock(C0, C1, oracle);

    byte[] plain;
    try {
        plain = removePadding(P1);
    } catch (Exception e) {
        plain = P1;
    }
    System.out.println(new String(plain, StandardCharsets.US_ASCII));
}
```

전반적인 입출력, 흐름을 제어하는 실행부 코드다. 중간에 발생할 수 있는 오류를 알리는 try-catch문을 사용했다.

4 Results

위의 구현을 통해 통해 아래와 같이 평문 복호화에 성공했다.

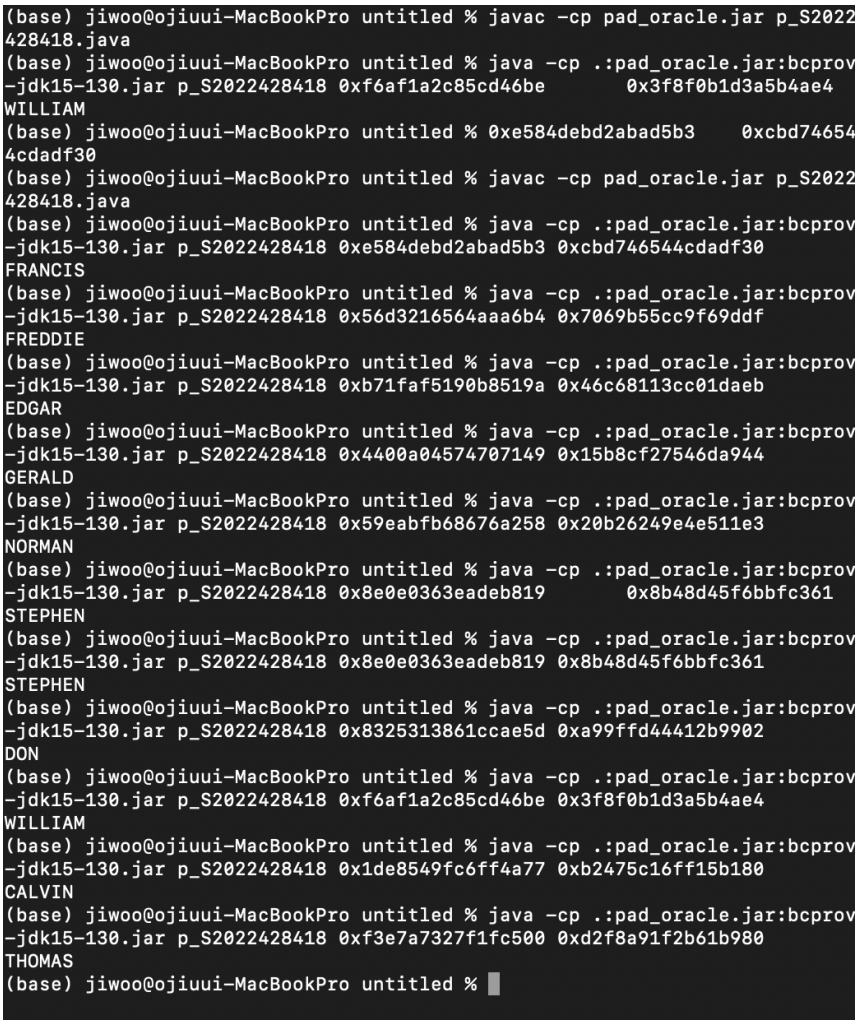

Figure 1: 결과(스크린샷)

실행 형식

```
javac -cp pad_oracle.jar p_S2022428418.java
java -cp .:pad_oracle.jar:bcprov-jdk15-130.jar p_S2022428418
0xe584debd2abad5b3(C0) 0xcbd746544cdadf30(C1)
```

최종 복호화 결과

```
ciphertext01.txt=FRANCIS
ciphertext02.txt=FREDDIE
ciphertext03.txt=EDGAR
```

```
ciphertext04.txt=GERARD
ciphertext05.txt=NORMAN
ciphertext06.txt=STEPHEN
ciphertext07.txt=DON
ciphertext08.txt=WILLIAM
ciphertext09.txt=CALVIN
ciphertext10.txt=THOMAS
```

5 Conclusion

5.1 목표에 대한 달성 여부

본 과제의 목표는 CBC 모드에서 발생하게 되는 Padding Oracle Attack을 구현해, 제공받은 패딩 오라클을 사용하여 단일 암호문 쌍으로부터 평문 블록 P를 복구하는 것에 있었다. 자세하게는 수업에서 제시되었던 vaudanay 공격을 직접 구현하여, 오라클에 반복적인 쿼리를 던져 암호문을 선택된 암호문 복호화 방식으로 공격하는 것이 목표였다. 결론적으로 이 과제의 구현 프로그램은 수업에서 제공된 패딩 오라클 인터페이스로 정상적으로 동작했으며, 주어진 ciphertext를 입력했을 때 평문을 ASCII로 출력하는 것을 볼 수 있어 구현에 성공했다. 구체적으로는 오른쪽부터 왼쪽으로 가며 바이트를 하나씩 찾는 알고리즘을 사용했고, 후보가 여러 개일 경우에도 백트래킹을 사용하여 다른 후보로 바꿔도록 하여 정확한 평문 출력에 성공했다.

5.2 발생 가능한 문제

프로그램 실행시 라이브러리 경로 문제와 같은 오류가 다수 발생했으므로 반드시 기준 환경에서 잘 동작할 수 있도록 채점 환경과 동일한 구조에서 실행해야 한다. 또한 오라클 호출이 값을 찾는 과정에서 백트래킹과 재시도로 많이 발생하여, 더 긴 블록에서는 실행 시간이 더 높아질 것으로 보여 개선이 필요하다.

5.3 시사점

본 과제의 성공은 CBC 모드에서 패딩 오류 정보만을 이용해 평문 복구가 가능함을 보인 사례다. 이를 통해 실무에서 우리가 오류 메시지와 검증 결과를 사용자들에게 노출시키는 것이 얼마나 큰 취약점이 될 수 있는지를 알 수 있었다. 시스템이나 보안 설계 시에는 이를 고려해 안전하게 설계하는 것이 필요할 것이라고 생각한다.