Padding Oracle 攻击实验

1 概述

本实验的学习目标是让学生亲身体验密码系统中的一种有趣攻击。有些系统在解密给定密文后，会检查填充是否有效；如果填充无效，系统就返回错误。这个看似无害的行为会造成一种被称为 *Padding Oracle 攻击*的漏洞。该攻击最早由 Serge Vaudenay 在 2002 年发表，后来许多知名系统都被发现曾受此类攻击影响，包括 Ruby on Rails、ASP.NET 和 OpenSSL。

在本实验中，我们提供两个运行在容器中的 oracle 服务器。每个 oracle 内部都隐藏了一条秘密消息，并会向你提供这条消息对应的密文和初始化向量（IV），但不会提供明文或加密密钥。你可以向 oracle 发送密文和 IV；oracle 会使用自己的密钥解密密文，并告诉你填充是否有效。你的任务是根据 oracle 的这些反馈推断出秘密消息的内容。本实验覆盖以下主题：

- 对称密钥加密
- 加密模式与填充
- Padding Oracle 攻击

阅读材料。 对称密钥加密的详细内容可以参考 SEED 教科书第 21 章：*Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`. 。当前版本的教材尚未详细覆盖 Padding Oracle 攻击，学生可以参考维基百科等在线资料了解攻击原理。

实验环境。 本实验在 SEED Ubuntu 20.04 VM 中测试可行。您可以从 SEED 网站上下载我们预先构建好的镜像并在您自己的电脑上运行 SEED VM。然而，大多数 SEED 实验可以在云端进行，您可以按照我们的说明在云端创建 SEED VM。

2 实验环境

在本实验中，我们使用容器运行 padding oracle。

容器设置和常用命令。 请从实验的网站下载 `Labsetup.zip` 文件到你的 VM 中,解压它,进入 `Labsetup` 文件夹，然后用 `docker-compose.yml` 文件安装实验环境。对这个文件及其包含的所有 `Dockerfile` 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的，那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

3 任务 1：熟悉填充机制

对于一些分组密码，如果明文长度不是分组长度的整数倍，就需要进行填充。PKCS#5 填充方案被许多分组密码广泛使用，详细内容可参考 SEED 教科书第 21.4 节。不过，PKCS#5 只为 8 字节分组定义；对于 AES 这类分组长度大于 8 字节的密码来说，这会带来问题。为了解决这个问题，人们提出了 PKCS#7 填充方案。本任务将通过实验帮助你理解这类填充是如何工作的。

我们可以使用 "echo -n" 命令创建文件。下面的例子创建了一个长度为 5 字节的文件 P。如果不使用 -n 选项，echo 会自动添加换行符，文件长度会变成 6 字节。

```
$ echo -n "12345" > P
```

我们使用 "openssl enc -aes-128-cbc -e" 对该文件进行加密，加密算法是 CBC 模式下的 128 位 AES。为了观察加密时补上的填充字节，我们会再用 "openssl enc -aes-128-cbc -d" 解密密文。默认情况下，解密命令会自动移除填充，因此我们无法直接看到填充内容。不过该命令提供了 "-nopad" 选项，用于禁止自动移除填充。这样，观察解密后的数据时，我们就能看到填充中使用了哪些字节。

```
$ openssl enc -aes-128-cbc -e -in P -out C
$ openssl enc -aes-128-cbc -d -nopad -in C -out P_new
```

需要注意，填充字节不一定是可打印字符，所以需要使用十六进制工具查看文件内容。下面的例子展示了如何用十六进制格式显示文件：

```
$ xxd P_new
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345..........
```

你的任务是分别创建长度为 5 字节、10 字节和 16 字节的三个文件。使用上面的方法，找出这三个文件在加密时分别添加了什么填充。

当解密 16 字节文件时，为什么会看到整整一个分组的填充？为什么这种做法是必要的？

4 任务 2：Padding Oracle 攻击（Level 1）

有些系统在解密给定密文时，会验证填充是否有效；如果填充无效，就抛出错误。这种看似无害的行为，会导致 *Padding Oracle* 攻击。该攻击最早由 Serge Vaudenay 于 2002 年发表，后来许多知名系统都被发现受到过这类攻击影响，包括 Ruby on Rails、ASP.NET 和 OpenSSL。

4.1 Oracle 设置

在本任务中，我们提供一个运行在 5000 端口上的 padding oracle。oracle 内部保存了一条秘密消息，并会打印出这条消息的密文。使用的加密算法和模式是 AES-CBC，加密密钥为 K，外部人员并不知道该密钥。你可以使用 "nc 10.9.0.80 5000" 与 oracle 交互。你会看到 oracle 提供的十六进制数据。前 16 个字节是 IV，其余部分是密文。从长度上看，密文有 32 字节，也就是 2 个分组；但由于存在填充，实际明文长度并不知道。

```
$ nc 10.9.0.80 5000
010203040506070801020304050607 08a9b2554b094411...
```

oracle 也接受你发送的输入。输入格式与上面的消息相同：16 字节 IV 后面拼接密文。oracle 会使用自己的密钥 K 和你提供的 IV 来解密密文。它不会告诉你明文，但会告诉你填充是否有效。你的任务是利用 oracle 提供的信息推断秘密消息的真实内容。为了简化问题，你只需要恢复秘密消息中的一个分组。为了调试方便，我们在下面给出了秘密消息，它也出现在 oracle 的源代码中。

```
static std::array<unsigned char, 29> PLAIN_TEXT = {
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
```

```
    0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
    0xaa, 0xbb, 0xcc, 0xdd, 0xee
};
```

这条秘密消息只用于调试，你不能假设自己已经知道它。你需要使用 Padding Oracle 攻击推导出这条消息，并在报告中展示推导过程。

4.2 手工推导明文

本任务的目标是找出秘密消息的明文。该消息包含两个分组 P1 和 P2。我们只需要恢复 P2，恢复 P1 的方法类似。实验材料中提供了一个名为 manual_attack.py 的代码框架。你可以在此基础上构造攻击。下面解释该代码的主要部分。

首先，从 oracle 获取密文。本任务中的密文由两个分组组成，我们使用两个 16 字节的 bytearray，即 C1 和 C2，保存这两个分组。

```
oracle = PaddingOracle('10.9.0.80', 5000)

# Get the IV + Ciphertext from the oracle
iv_and_ctext = bytearray(oracle.ctext)
IV    = iv_and_ctext[00:16]
C1    = iv_and_ctext[16:32]  # 1st block of ciphertext
C2    = iv_and_ctext[32:48]  # 2nd block of ciphertext

print("C1:  " + C1.hex())
print("C2:  " + C2.hex())
```

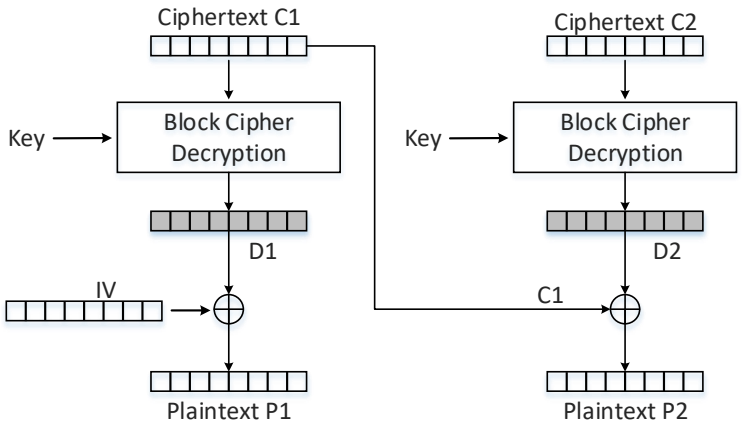

图 1: CBC 模式下的解密过程

如图 1 所示，D1 和 D2 是 AES 分组密码的输出。如果能够得到它们的值，就可以通过与密文分组（或第一个分组对应的 IV）异或来得到明文。本任务重点关注第二个分组 P2。因此，只要能得到 D2，就可以通过 P2 = C1 xor D2 计算出 P2。

本实验说明不会完整推导 Padding Oracle 攻击的数学细节，学生可以参考维基百科等资料。攻击的大致思路是：向 oracle 发送一个修改过 C1 的密文分组，这里称为 CC1。oracle 虽然不会告诉我们 D2 xor CC1 的结果，但它会告诉我们该结果的填充是否有效。这个反馈就为推断 D2 打开了入口。

D2 一共有 16 个字节，我们可以一次恢复一个字节。在代码框架中，我们初始化了 D2 和 CC1 两个数组。初始值并不重要，你的任务是通过迭代过程找出 D2 的正确值。每轮迭代中，你都需要正确构造 CC1 数组。

```
# The initial value of D2 does not matter. Our job is to
# find its correct values.
D2 = bytearray(16)

D2[0]  = C1[0]
D2[1]  = C1[1]
...
D2[15] = C1[15]

# CC1 is used to replace the C1 block in the ciphertext
# Its values need to be set properly in each round
CC1 = bytearray(16)

CC1[0]  = 0x00
CC1[1]  = 0x00
...
CC1[15] = 0x00
```

在每轮迭代中，我们只关注 CC1 的一个字节。我们尝试这个字节的 256 种可能值，把构造出的密文 CC1 + C2（再加上 IV）发送给 oracle，并观察哪个值能让填充有效。只要构造正确，就会有一个值满足条件。这个值可以帮助我们恢复 D2 的一个字节。下面的代码关注 K=1 的情况，它可以帮助我们找出 D2[15] 的值。

```
K = 1
for i in range(256):
      CC1[16 - K] = i
      status = oracle.decrypt(IV + CC1 + C2)
      if status == "Valid":
          print("Valid: i = 0x{:02x}".format(i))
          print("CC1: " + CC1.hex())
```

你可以基于该代码框架，手动改变 K 的值；利用每一轮运行结果设置相应的 D2 字节；然后用更新后的 CC1 重新运行程序，继续恢复下一个字节，例如 D2[14]。重复该步骤，就可以恢复 D2[13]、D2[12]，直到 D2[0]。一旦获得完整的 D2，就能得到明文 P2。

```
# Once you get all the 16 bytes of D2, you can easily get P2
P2 = xor(C1, D2)
print("P2:  " + P2.hex())
```

说明。 恢复 D2 的全部 16 个字节可能比较繁琐。学生可以在恢复 6 个字节后停止，这足以得到 P2 最后 6 个字节（包括填充）。虽然完全可以写程序自动完成整个过程，但本任务的目的就是让学生手工完成攻击步骤。因此，实验报告中需要包含至少 6 个字节的 D2 是如何得到的。下一个任务会要求学生自动化该过程。

5 任务 3：Padding Oracle 攻击（Level 2）

在上一个任务中，我们进行了手工攻击。本任务将自动化整个攻击过程，并恢复明文的所有分组。容器启动时会启动两个 padding oracle 服务器：一个用于 Level-1 任务，另一个用于本任务。Level-2 服务器监听 6000 端口。虽然密钥和秘密消息位于 oracle 程序的二进制文件中，我们已经对它们做了混淆，因此并不容易直接从二进制文件中找到。更重要的是，知道秘密消息本身并不能帮助理解 Padding Oracle 攻击。评分依据是你如何使用 Padding Oracle 攻击推导秘密消息，而不是你是否事先知道消息内容。

需要注意，每次你与 oracle 建立新连接时，oracle 都会生成新的密钥和 IV 来加密秘密消息（消息内容本身不变），所以你会看到不同的密文。不过，如果你一直保持在同一个连接中，密钥和 IV 不会改变。

你可以编写程序，在一次运行中恢复秘密消息的所有分组；也可以让程序一次恢复一个分组。最终，你需要打印出所有分组的内容。在报告中，请提交你的代码以及程序运行结果的截图。

6 提交要求

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。

7 致谢

感谢以下人员和机构对本实验的贡献：

- Jiamin Shen 开发了容器内部运行的代码，以及 Padding Oracle 攻击任务的初始版本。
- 美国国家科学基金会在 2002 年至 2020 年期间为 SEED 项目提供资金支持。
- Syracuse University 自 2001 年以来为 SEED 项目提供资源支持。