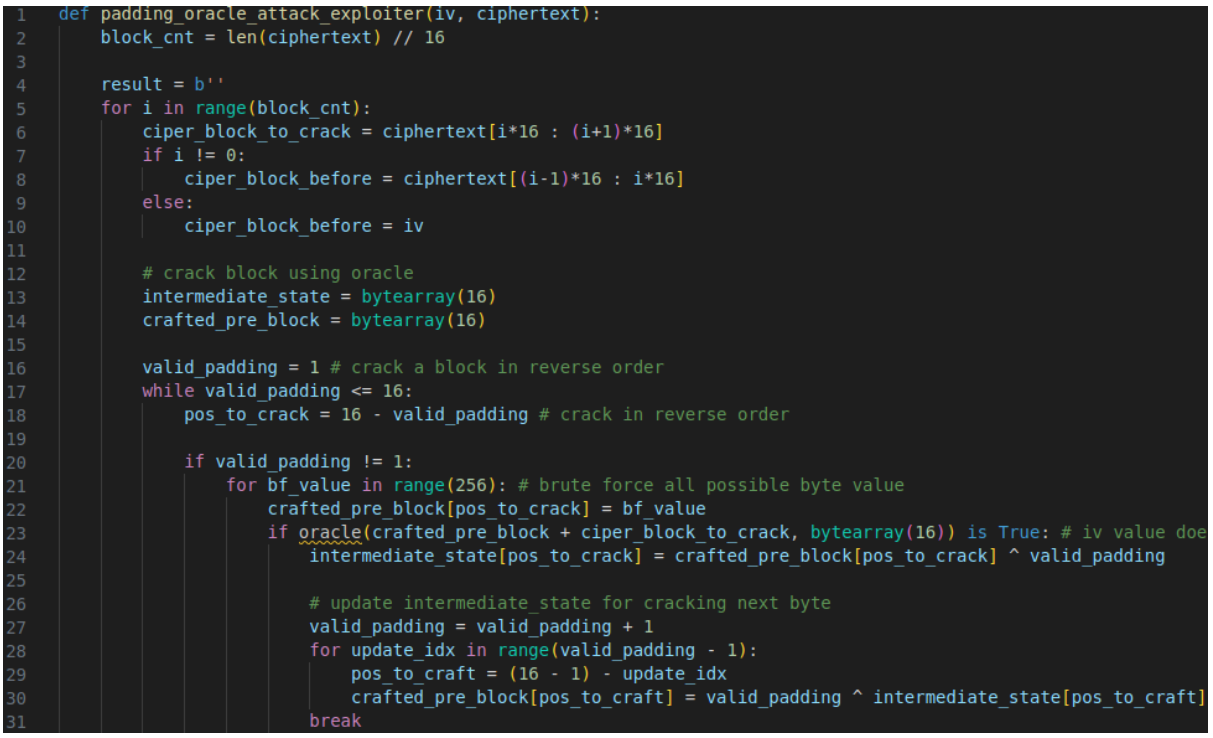
CS528 Lab 1 Report
Zheng Zhong (zhong183)

Padding Oracle Attack

The padding oracle attack is an attack to recover plaintext from ciphertext for any block cipher with PKCS7 padding scheme and CBC mode, by passing crafted ciphertext to an oracle which returns whether the decrypted message has valid padding or not.

Implementation



```
1  def padding_oracle_attack_exploiter(iv, ciphertext):
2      block_cnt = len(ciphertext) // 16
3
4      result = b''
5      for i in range(block_cnt):
6          ciper_block_to_crack = ciphertext[i*16 : (i+1)*16]
7          if i != 0:
8              ciper_block_before = ciphertext[(i-1)*16 : i*16]
9          else:
10             ciper_block_before = iv
11
12         # crack block using oracle
13         intermediate_state = bytearray(16)
14         crafted_pre_block = bytearray(16)
15
16         valid_padding = 1 # crack a block in reverse order
17         while valid_padding <= 16:
18             pos_to_crack = 16 - valid_padding # crack in reverse order
19
20             if valid_padding != 1:
21                 for bf_value in range(256): # brute force all possible byte value
22                     crafted_pre_block[pos_to_crack] = bf_value
23                     if oracle(crafted_pre_block + ciper_block_to_crack, bytearray(16)) is True: # iv value doe
24                         intermediate_state[pos_to_crack] = crafted_pre_block[pos_to_crack] ^ valid_padding
25
26                         # update intermediate_state for cracking next byte
27                         valid_padding = valid_padding + 1
28                         for update_idx in range(valid_padding - 1):
29                             pos_to_craft = (16 - 1) - update_idx
30                             crafted_pre_block[pos_to_craft] = valid_padding ^ intermediate_state[pos_to_craft]
31                         break
```

Function padding_oracle_attack_exploiter is a working implementation of poa attack. Line 2 computes the number of blocks according to the length of ciphertext. Line 5-10 iterate through each block. For each block, line 16-24 crack each byte in reverse order. Line 21-23 enumerate all possible values of a byte, set specific location in the crafted block and concatenate crafted block with ciphertext block, query the padding oracle until it returns true to obtain the valid padding value in the crafted plaintext. Line 24 computes a corresponding byte of intermediate state using valid padding value and crafted block value. Line 27-31 set the cracked bytes of crafted block value for the next padding value querying round.

```
32              else:  # special handling for first padding value
33                  oracle_true_bf_value = 0
34                  oracle_true_cnt = 0
35                  for bf_value in range(256): # brute force all possible byte value
36                      crafted_pre_block[pos_to_crack] = bf_value
37                      if oracle(crafted_pre_block + cipher_block_to_crack, bytearray(16)) is True: # iv value
38                          oracle_true_cnt += 1
39                          oracle_true_bf_value = bf_value
40
41                  if oracle_true_cnt == 1:
42                      crafted_pre_block[pos_to_crack] = oracle_true_bf_value
43                      intermediate_state[pos_to_crack] = crafted_pre_block[pos_to_crack] ^ valid_padding
44
45                      # update intermediate state for cracking next byte
46                      valid_padding = valid_padding + 1
47                      for update_idx in range(valid_padding - 1):
48                          pos_to_craft = (16 - 1) - update_idx
49                          crafted_pre_block[pos_to_craft] = valid_padding ^ intermediate_state[pos_to_craft]
50
51                  else: # more than one value have true oracle return value, craft another pre block and try
52                      crafted_pre_block[pos_to_crack - 1] += 1
```

Line 32-52 handles the special case for the first valid padding value 1. For the first padding value, the padding oracle may return true if the decrypted text ends with "01", "02, 02", "03, 03" and so on, because none of the decrypted text bytes is determined. We use a counter to count the number times the oracle returns true over all 256 possible last byte value. If the count is one, the decrypted text must end with "01" because "01" is simply a valid padding. In this case we handle everything as the normal cases (line 42-49). Otherwise we change the second last byte value of the crafted block and try again (line 51-52).

```
55          # reconstruct plaintext: plaintext_block_this = ciper_block_before ^ intermediate_state
56          plain_block_exploit = bytearray(16)
57          for p_idx in range(16):
58              plain_block_exploit[p_idx] = ciper_block_before[p_idx] ^ intermediate_state[p_idx]
59          result =  result + plain_block_exploit
60
61      return result
```

After recovering all bytes of intermediate state, line 55-59 compute plaintext using the equation plain_block_exploit = ciper_block_before ^ intermediate_state.

Result

```
(normal) syssec@syssec-NUC7i7BNH:~/hw/cs528/lab1$ python poa.py
b'This is cs528 padding oracle attack lab with hello world~~~!!'
<class 'bytes'>
padding_oracle_attack_exploiter recover message: b'This is cs528 padding oracle
attack lab with hello world~~~!!\x03\x03\x03'
(normal) syssec@syssec-NUC7i7BNH:~/hw/cs528/lab1$
```

The above screenshot shows the padding_oracle_attack_exploiter successfully recovers the plaintext with 3 padding bytes \x03\x03\x03 at the end. To test other messages, simply change the plaintext value in the __main__ function of poa.py.