Computer Security Hw0x09 Writeup

- Realname: 胡安鳳
- ID on course web: alfons0329
- Student ID: R08922024

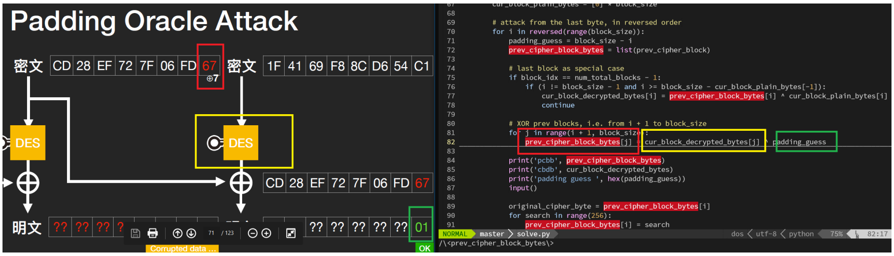
tags: `Computer Security` `NTU CS` `CS` `CTF` `Writeup`

CatHub Party (crypto)

Padding Oracle Attack

- The step of implementing this attack can be found in this video (https://www.youtube.com/watch?v=vzW37CjEJgs), timestamp = 1hr15min ~ 1hr21min
- And the well-explained wikipedia page here (https://en.wikipedia.org/wiki/Padding_oracle_attack)

Step 1: XOR the adjacent 2 blocks of cipher for the prev block.



```
    cur_block_plain_bytes = [0] * block_size

    # attack from the last byte, in reversed order
    for i in reversed(range(block_size)):
        padding_guess = block_size - i
        prev_cipher_block_bytes = list(prev_cipher_block)

        # last block as special case
        if block_idx == num_total_blocks - 1:
            if (i != block_size - 1 and i >= block_size - cur_block_plain_bytes[-1]):
                cur_block_decrypted_bytes[i] = prev_cipher_block_bytes[i] ^ cur_block_plain_bytes[i]
                continue

        # XOR prev blocks, i.e. from i + 1 to block_size
        for j in range(i + 1, block_size):
            prev_cipher_block_bytes[j] = cur_block_decrypted_bytes[j] ^ padding_guess

        print('pcbb', prev_cipher_block_bytes)
        print('cbdb', cur_block_decrypted_bytes)
        print('padding guess ', hex(padding_guess))
        input()

        original_cipher_byte = prev_cipher_block_bytes[i]
        for search in range(256):
            prev_cipher_block_bytes[i] = search
NORMAL  master > solve.py                                 dos < utf-8 < python   75%   82:17
/\<prev_cipher_block_bytes\>
```

The mathematical formula for CBC decryption is

$$P_i = D_K(C_i) \oplus C_{i-1}$$
$$C_0 = IV$$

Given the example for current round's padding guessing.
Suppose, in this round, the padding we are trying to guess being $0x01$.

We now knows that the last byte of $D_K(C_2) \oplus C_1'(Last\ byte\ of\ cipher\ block\ 1)$ is $0x01$
Therefore, $D_K(C_2) = C_1' \oplus 0x01$ and $C_1' = D_K(C_2) \oplus 0x01$, describing what has been done in the picture shown above.

Step 2: Brute force searching for the corresponding byte at previous block.

```
original_cipher_byte = prev_cipher_block_bytes[i]
for search in range(256):
    prev_cipher_block_bytes[i] = search
    guess_flag_cipher = b''.join(flag_cipher_blocks[0: block_idx - 1]) \
    + bytes(prev_cipher_block_bytes) \
    + flag_cipher_blocks[block_idx]

    # the padding is correct,
    # no need to guess anymore (try to match correctly with padding)
    if interact_with_site(session, \
    urlencode(base64.b64encode(guess_flag_cipher).decode())) \
    == WRONG_FLAG_CORRECT_PADDING:
        cur_block_decrypted_bytes[i] = search ^ padding_guess
        cur_block_plain_bytes[i] =  original_cipher_byte \
        ^ cur_block_decrypted_bytes[i]
        break
```

If the previous-mentioned padding is correct (i.e. directly guessed the padding correctly), then we now know that the last byte of $D_K(C_2) \oplus C_1'$ is $0\text{x}01$.
Therefore, $D_K(C_2) = C_1' \oplus 0\text{x}01$. --> `cur_block_decrypted_bytes[i] = search ^ padding_guess`
And for plain text $P_2 = D_2(C_2) \oplus C_1$ --> `cur_block_plain_bytes[i] = cur_block_decrypted_bytes[i] ^ original_cipher_byte`

If the padding is incorrect, the we can change the last byte of C_1' to the next possible value `prev_cipher_block_bytes[i] = search`, just keep brute force searching.

At most, the we will need to make 256 attempts (one guess for every possible byte) to find the last byte of P_2

- The rest of the exploitation code and explanation are in `solve.py` of this homework.