Network Administration HW7

B04705003 資工三 林子雋

1 Reference

1.1 Cryptolocker

1. cryptolocker Writeup
2. https://timothybramlett.com/Strings_Bytes_and_Unicode_in_Python_2_and_3.html

1.2 2AES

1. https://www.wikiwand.com/en/Meet-in-the-middle_attack

1.3 Man in the Middle 2

1. https://www.thesecuritybuddy.com/dos-ddos-prevention/authentication-reflection-atta

2 Problem

2.1 Cryptolocker

1. Flag: `Ev3n_s3cure_PRF_1s_bre4k4ble_1f_u5ed_unc0rr3ctly`
2. Explanation: If using brute force, we need to search 8 bytes keys and it is impossible to search such a large number by naive brute force. Therefore, we need to do some trick to reduce search space when searching keys. The trick is that in `AESCipher.py`, before it encrypts raw data, it first pads the data by `self.bs - len(s) % self.bs` and use these padding bytes to represent how many padding bytes are behind original data. Therefore, while brute force decoding, we can only leave those immediate cipher which are valid padded.
3. Script: See `cryptolocker_decode.py`

2.2 2AES

1. Flag: `Tw0_3qual5_to_0ne_Whyyyyyy`

2. Explanation: Because we already have a pair of plaintext and cipher, we can perform meet-in-the-middle attack. First, enumerate all the possible keys(say, key zero) and save its corresponding intermediate cipher by python hash dictionary. Then, enumerate all the possible keys(say, key one) to see if there is an intermediate cipher decrypted from the final cipher matches the intermediate cipher that we previous generate. If there is, that key pair is what we want.

3. Script: See `2AES_mimt.py` and `2AES_mimt_decode.py`

2.3 Man in the Middle 2

1. Flag: `Ahhhh...I_g0t_Mitm_4g41n...`

2. Explanation: Because in `mimt.py`, we see that the server use XOR to encode the flag, we can try to exploit this vulnerability. First, we create two connections to the server. If we want to guess the first round's g, we can fixed a and send $g^a \bmod p$ to the server in the first round. While in other rounds, we exchange the incoming B and send them to different connection respectively. In the end, we can use the property of XOR and verify that if our guess in the first round is correct. So on so forth, we can figure out g in every round. See Figure 1 for the illustration.

3. Script: See `search_password.py`

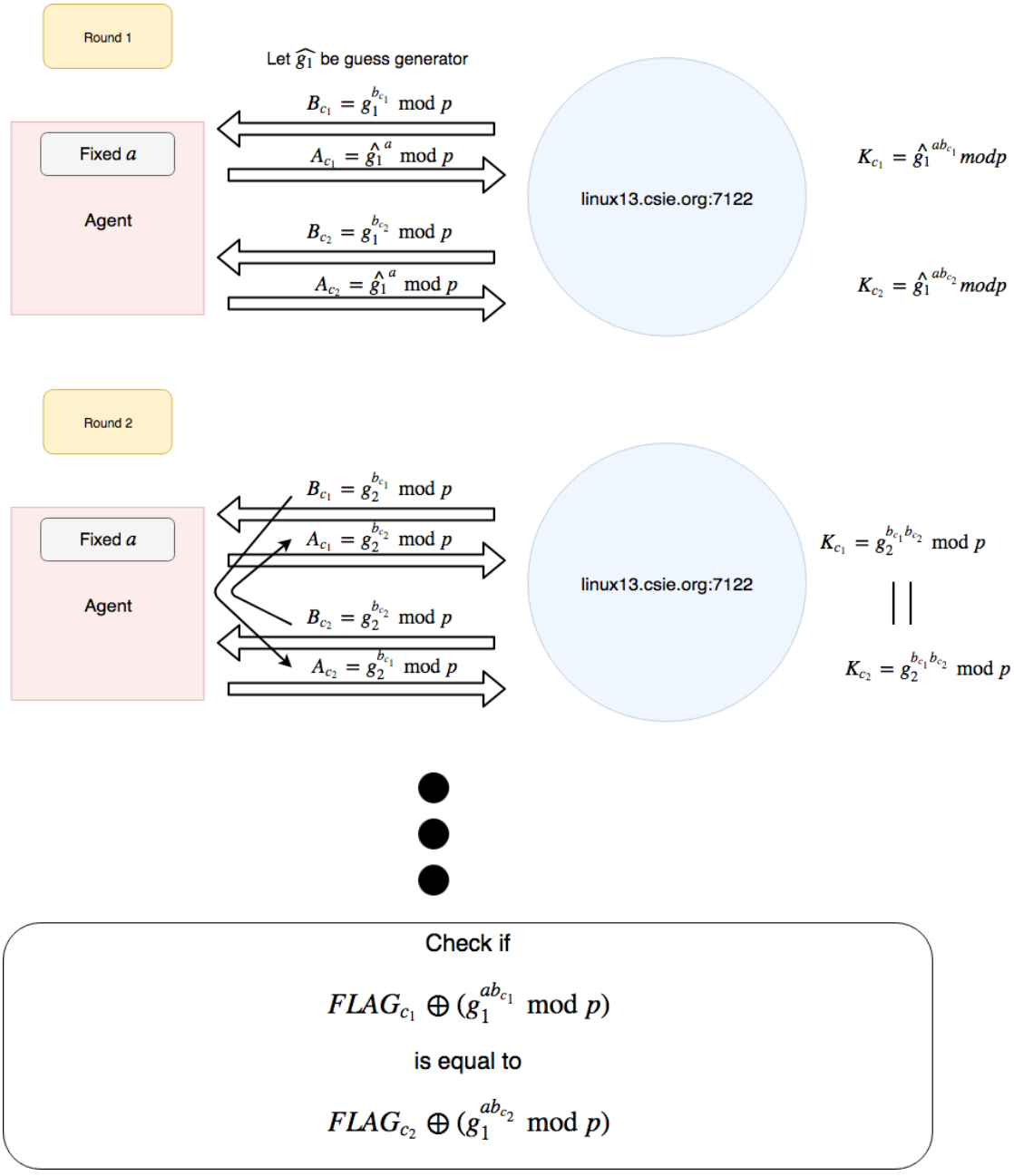

Figure 1: Reflection Attack Workflow