**Deliverables:**

FOLLOW THE ACTIVITY TEMPLATE LOCATED IN COURSE INFORMATION SECTION OR YOUR ACTIVITY WILL BE PENALIZED!

**I.             Cover Sheet (5 Points)**

Complete the cover sheet, including name, title, summary, key results, key conclusions (technical), and key conclusions (critical thinking).

**II.          Introduction (5 Points)**

A short (one paragraph) introduction that identifies the objectives of this activity.  Summarize the case study associated with this activity.

**III.       Procedure (5 Points)**

Discuss the steps involved in working through the activity.  Summarize and discuss the process that is being used to highlight the concepts associated with this activity.

**IV.       Results: Data, Tables, & Diagrams (40 Points)**

Provide the answers to the questions related to the activity.  Include any documentation or screenshots relevant to the activity.

**V.           Analysis of Results (10 Points)**

Write a short analysis that highlights the important characteristics of the activity.  Include your analysis on how this information relates to Network Forensics.  Discuss how the tools used allowed you to better understand the concepts related to this activity and Network Forensics.

**VI.       Summary and Conclusions (10 Points)**

Summarize this activity.  Include key aspects of the activity.  Include what you learned during this exercise and describe how it helped you to better understand the Network Forensic Concepts presented in the chapter lectures and readings.

**Module 5 Activity:**

##### [**Case Study: HackMe,**](#_bookmark2) **Inc.[[1]](#footnote-1)**

The Case: September 17th, 2010: Inter0ptic is on the lam and is pinned down. The area is crawling with cops, and so he must stay put. But he also desperately needs to be able to get a message out to Ann and Mr. X. Lucky for him, he detects a wireless access point (WAP) in the building next door that he might be able to use. But it is using encryption, and there are no other opportunities available. What is Inter0ptic to do?

Meanwhile ... Next door, Joe is a sysadmin at HackMe, Inc. He runs the technical infrastructure for a small company, including a WAP that is used pretty much exclusively by him. He’s trying to use it now, and has discovered that he’s begun to get dropped. He captures some traffic, but he really has no idea how to interpret it. Suddenly he discovers he can’t even login to administer his WAP at all!

The Challenge: You are the forensic investigator. Your team got a tip that Inter0ptic might be hunkered down in the area. Can you ﬁgure out what’s going on and track the attacker’s activities?

The following questions will help guide your investigation:

What are the BSSID and SSID of the WAP of interest?

Is the WAP of interest using encryption?

What stations are interacting with the WAP and/or other stations on the WLAN?

Are there patterns of activity that seem anomalous?

How are they anomalous: Consistent with malfunction? Consistent with maliciousness?

Can we identify any potentially bad actors?

Can we determine if a bad actor successfully executed an attack?

Evidence: Joe has provided you with a packet capture (wlan.pcap) and permission to inspect it in any way you need to either solve his problem, catch Inter0ptic, or both. He also helpfully tells you that his own system’s MAC address is 00:11:22:33:44:55, and reiterates that no one else should be using his WAP.

**Inspecting the WAP**

The most obvious place to begin analysis is Joe’s WAP. Along the way we expect—or at least hope—to learn something about the stations with which it was communicating, and to be able to infer a whole lot from the anomalous traﬃc we’re about to examine. Let’s begin by identifying and inspecting the WLAN under investigation. Change your directory to Module\_5\_Files in your Documents folder in the terminal CLI.

**Inspecting Beacon Frames**

Probably the most straightforward way to identify the WAPs in a packet capture is to simply ﬁlter on Beacon frames. Wireshark can be used with a display ﬁlter on the appropriate frame type (0) and subtype (8): “wlan.fc.type\_subtype == 0x08.” Note also the “BSS Id” in the frame: 00:23:69:61:00:d0. Go to your Documents folder in the Virtual Box application and open the pcap file. Enter the filter:

wlan.fc.type\_subtype= =0x80

The filter window should turn green and then you apply the filter.

1. Capture a screenshot of your result.

Filter on WAP-Announcing Management Frames

Let’s use our tshark invocation to ﬁlter traﬃc and display only Beacon and Probe Response frames that have the ESS subﬁeld set to 1 and the IBSS subﬁeld set to 0. (Recall that, by speciﬁcation, WAPs set these ﬁelds accordingly.) Even if a WAP is not broadcasting Beacon frames, it may still send Probe Responses to stations that initiate Probe Requests.

$ tshark -nn -r wlan . pcap -Y '(( wlan .fc. type\_subtype == 0x08 || wlan .fc. type\_subtype == 0x05) && ( wlan\_mgt . fixed. capabilities.ess == 1) && ( wlan\_mgt . fixed. capabilities. ibss == 0)) '

1. Capture a screenshot of your result.

You could also use the same display filter in Wireshark to find Beacon or Probe Response frames that emanate from the WAP.

1. Capture a screenshot of the Wireshark result and compare the tshark and Wireshark outputs.

To list the BSSIDs of the known WAPs in the packet capture, let’s use tshark along with some simple shell scripting. In the command below, we tell tshark to print only the BSSID ﬁeld (“wlan.bssid”), and then we use the Linux tool “uniq” to get a count of the number of occurrances of each BSSID.

$ tshark -nn -r wlan . pcap -Y '(( wlan .fc. type\_subtype == 0x08 || wlan .fc. type\_subtype == 0x05) && ( wlan\_mgt . fixed. capabilities.ess == 1) && ( wlan\_mgt . fixed. capabilities. ibss == 0)) ' -T fields -e wlan . bssid | uniq -c

1. How many WAPs sent Beacon or Probe Response frames in the packet capture?
   1. How many frames did it send?
   2. What is the BSSID?

**WLAN Inventory**

Returning to the Beacon frame from earlier, let us inspect it further with Wireshark so that we can record a few more details about the WLAN that we may need later (and perhaps even create some display ﬁlters we can use later on with “tshark” to speed things up). Along the way we’ll answer a number of the questions posed in the challenge.

Enter the filter information ((wlan.fc.type\_subtype == 0x08 || wlan.fc.type\_subtype == 0x05) && (wlan\_mgt . fixed. capabilities.ess == 1) && ( wlan\_mgt . fixed. capabilities. ibss == 0)) into Wireshark

1. Capture a screenshot of your output.
   1. What is the SSID? (Hint: Look in the management frame – subtype 0x8)
   2. On which channel is WAP operating

**Encryption**

1. Is the WLAN using encryption?
   1. Capture a screenshot of your output

To answer this question, ﬁrst, we can ask Wireshark to show us only the data frames: “wlan.fc.type subtype == 0x20.”

Using tcpdump and some BPF inspection, we can easily demonstrate that *all* the data frames in the capture are WEP protected. First, let’s count the data frames. By ﬁltering on the version (0), type (2) and subtype (0) in the ﬁrst byte of the frame (the byte at “oﬀset zero”), and sending the output to the “wc” command to count lines, we get 59,274 total data frames: (Note: wc -l is wc “lower case L” and not “1” one)

$ tcpdump -nne -r wlan . pcap ' wlan [0] = 0x08 ' | wc -l

1. Capture a screenshot of your result.
   1. What was the value?

Next, let’s also ﬁlter on the “Protected” bit in the next byte (again, keeping in mind the endianness of the bits in transmission):

$ tcpdump -nne -r wlan . pcap ‘wlan [0] = 0x08 and wlan [1] & 0x40 = 0x40’ | wc -l

1. Capture a screenshot of this result.
   1. Are the values the same?
   2. What does this tell you?

**Associated Stations**

Using Wireshark, we can easily construct a ﬁlter on the Association Response subtype of the management frames, and further ﬁlter on the 2-byte status code indicating successful association: “wlan.fc.type subtype == 0x01 && wlan mgt.ﬁxed.status code == 0x0000.” Presumably the source of such frames should all be our known BSSID, and the various destinations would be the stations that successfully associated.

1. Capture a screenshot of your Wireshark output.

With a little bit of tcpdump and BPF language, and some more command-line entries, we can aggregate data from these frames in a useful way. We begin by ﬁltering on the version/type/subtype byte for the appropriate value for an Association Response (0x01, which translates to 0x10 in transmission order), and then locate the 2-byte ﬁeld that indicates the status code (the 2 bytes starting at the 26th byte oﬀset (remember, counting from 0)).

In the tcpdump output produced by the command below, the destination MAC address is the third ﬁeld. We use the “awk” command to print only the third ﬁeld, and then send these values line-by-line to “sort,” whose output then allows the “uniq” program to aggregate and count them. By sending the remaining output through the last “sort” invocation, we can see them listed in order of frequency.

$ tcpdump -nne -r wlan . pcap ' wlan [0] = 0x10 and wlan [26:2] = 0x0000 '| awk '{ print $3}'| sort| uniq -c| sort -nr

reading from file wlan .pcap , link - type IEEE802\_11 (802.11)

68 DA :1 c:4 b: d6 :69: cd:07

4 DA :00:11:22:33:44:55

1 DA : de: ad: be: ef :13:37

It seems that Joe successfully associated four times, an unknown station with the MAC address of “de:ad:be:ef:13:37” associated once, and another unknown station with the MAC address of “1c:4b:d6:69:cd:07” successfully associated 68 times. That seems a bit odd.

**Statistics**

Let’s gather some statistics on the traﬃc in this packet capture in order to better understand the actors and time frame.

Let’s look again at the encrypted data frames. How many encrypted data frames are there? Who are they coming from and where are they going? Are there any that seem unusual?

Tshark is an excellent tool to help us look at statistics, as well as individual packets. Let’s begin by trying out a display ﬁlter that should provide a known result: the number of protected data frames in the capture. Since we know the BSSID of the WAP that is the focus of our investigation, we will include the ﬁlter “wlan.bssid == *enter 6 byte value from earlier here – format xx:xx:xx:xx:xx*” in our tshark invocations in order to narrow the scope to our identiﬁed target of interest.

Note that in the display ﬁlter below, we specify the value of the type/subtype ﬁelds as the appear in the protocol speciﬁcation (0x20), NOT the order in which they are transmitted (0x08). This is an important diﬀerence between the display ﬁlters we use with tshark/Wireshark and the BPF ﬁlters we used previously with tcpdump.

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == *enter 6 byte value from earlier here*)'|wc -l

1. Capture a screenshot of your result.

The number should correspond with the results of our previous inspection. From there, we can begin to use tshark to extract individual ﬁelds from the WLAN protocol for aggregation and comparison. This invocation, the end of which should be familiar now, shows us an aggregated count of the number of encrypted data frames transmitted from each MAC address:

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == *enter 6 byte value from earlier here*) ' -T fields -e wlan . sa| sort| uniq -c| sort -nr

In the example above, by extracting only the “sending address” ﬁeld of the WLAN protocol (“wlan.sa”) from the data frames, and sorting and counting their occurrences, we can see that one of our unknown stations (1c:4b:d6:69:cd:07) sent roughly three times the number of data frames as Joe’s station during the same time period.

We also see a source of “00:23:69:61:00:ce,” which diﬀers from the WAP’s BSSID by only the last octet. It’s interesting that this MAC address is very similar to that of the WAP’s BSSID. Recall that WAPs typically serve at least two functions: First, they provide access to wireless distribution services, and second, they act as stations on the network which provide services for WAP management, DHCP, logging, and other functionality. It is common to see diﬀerent addresses used for diﬀerent purposes. In this case, let’s hypothesize that the similarity between the MAC address “00:23:69:61:00:ce” and the WAP’s BSSID is not a coincidence. It is likely that “00:23:69:61:00:ce” is the WAP’s MAC address which it uses to participate as a logically distinct station in the wireless network. We will henceforth refer to this informally as the WAP’s “station” (STA) interface.

Last but not least, we also see a few data frames from the odd MAC address, “de:ad:be:ef:13:37.”

What if we were to look at the comparative count of frame destination addresses (“wlan.da”) in the same fashion?

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == 00:23:69:61:00: d0 ) ' -T fields -e wlan . da| sort| uniq -c| sort -nr

1. Capture a screenshot of the result.

42837 ff: ff: ff: ff: ff: ff

14816 00:23:69:61:00: ce

858 00:11:22:33:44:55

654 de: ad: be: ef :13:37

59 01:00:5 e:7f:ff:fa

25 33:33:00:00:00:02

17 33:33:00:00:00:16

6 33:33: ff :33:44:55

2 33:33: ff: ef :13:37

Interesting: The results shown above indicate that approximately 42,837 encrypted data frames were sent to the broadcast MAC address (ﬀ:ﬀ:ﬀ:ﬀ:ﬀ:ﬀ). This is almost the same as the number of frames sent from station 1c:4b:d6:69:cd:07. Similarly, station 00:23:69:61:00:ce (likely the WAP’s STA interface) received roughly the same number of encrypted data frames as Joe’s station (00:11:22:33:44:55). Coincidence?

Let’s look at both source and destination:

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == 00:23:69:61:00: d0 ) ' -T fields -e wlan . sa -e wlan.da| sort| uniq -c| sort -nr

42816 1c:4 b: d6 :69: cd :07 ff: ff: ff: ff: ff: ff

14076 00:11:22:33:44:55 00:23:69:61:00: ce

858 00:23:69:61:00: ce 00:11:22:33:44:55

740 de: ad: be: ef :13:37 00:23:69:61:00: ce

654 00:23:69:61:00: ce de: ad: be: ef :13:37

59 00:23:69:61:00: ce 01:00:5 e:7f:ff:fa

18 00:11:22:33:44:55 33:33:00:00:00:02

14 00:11:22:33:44:55 ff: ff: ff: ff: ff: ff

13 00:11:22:33:44:55 33:33:00:00:00:16

7 de: ad: be: ef :13:37 33:33:00:00:00:02

6 00:11:22:33:44:55 33:33: ff :33:44:55

4 de: ad: be: ef :13:37 ff: ff: ff: ff: ff: ff

4 de: ad: be: ef :13:37 33:33:00:00:00:16

3 00:23:69:61:00: ce ff: ff: ff: ff: ff: ff

2 de: ad: be: ef :13:37 33:33: ff: ef :13:37

Sure enough, that mystery station (1c:4b:d6:69:cd:07) sent all 42,816 of its data frames to the broadcast address. Most of the remaining frames were sent between Joe’s station and the WAP’s presumed STA interface (with the majority of those frames sent from Joe’s station). The next most common exchanges appear to be between the other odd station, “de:ad:be:ef:13:37,” and the WAP.

* 72% were sent from an unknown station (1c:4b:d6:69:cd:07) to the broadcast address (ﬀ:ﬀ:ﬀ:ﬀ:ﬀ:ﬀ)
* 25% were transmitted between Joe’s station (00:11:22:33:44:55) and the WAP (00:23:69:61:00:ce)
* 2% were transmitted between an unknown station (de:ad:be:ef:13:37) and the WAP (00:23:69:61:00:ce)

There are relatively few reasons that data frames would be sent to the broadcast MAC address. Perhaps the most common case is when clients send out ARP requests. However, the volume of traﬃc sent by 1c:4b:d6:69:cd:07 compared with other stations seems abnormally high to be legitimate ARP requests.

Another possibility is that 1c:4b:d6:69:cd:07 was conducting some sort of attack on the wireless network. WEP-cracking attacks often involve the attacker sending out a large number of 802.11 data frames. Recall that an attacker’s ability to leverage the related key attack depends on the volume of unique IVs exposed. In order to capture lots of unique IVs quickly, the attacker needs to send out traﬃc that triggers other stations on the network to respond. An eﬀective tactic is to replay ARP requests, because ARP requests elicit timely responses from other systems.

A malicious actor can capture 802.11 frames and replay them on the network, even without knowing the WEP key. How does an attacker know which 802.11 traﬃc to replay if the traﬃc is encrypted? In the classic ARP replay attack, the attacker listens for data frames sent to the broadcast MAC address—likely ARP requests—and just blindly replays them. In this way, the attacker has a good chance of generating traﬃc that will trigger a response from other stations on the WLAN. By replaying data packets to the broadcast Layer 2 address, an attacker can cause other stations to generate frames with unique IVs, which can then be captured and used to dramatically speed up the WEP-cracking attack.

**Patterns and Time Frames**

Are there any patterns or stations that seem unusual (perhaps considering the known vs. unknown stations by volume)? Let’s use the Wireshark suite’s “capinfos” tool to quickly view the duration of the packet capture and other high-level information.

$ capinfos wlan . pcap

1. Capture a screenshot of your result

For more granular timestamps, we can use tcpdump to print the ﬁrst and last frame:

$ tcpdump -nnr wlan . pcap| head -1

$ tcpdump -nnr wlan . pcap| tail -1

1. What is the approximate total duration of the packet capture?

In that time frame, Joe’s station (11:22:33:44:55:66) sent the following data frames to the destinations shown below:

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 00:11:22:33:44:55) ' -T fields -e wlan .da| sort| uniq -c| sort -nr

14076 00:23:69:61:00: ce

18 33:33:00:00:00:02

14 ff: ff: ff: ff: ff: ff

13 33:33:00:00:00:16

6 33:33: ff :33:44:55

1. In the same amount of time, how many data frames did the unknown station (1c:4b:d6:69:cd:07) send to broadcast?
   1. Capture your screenshot.
   2. Were any frames sent to a different address?

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 1c:4 b

:d6 :69: cd :07) ' -T fields -e wlan .da| sort| uniq -c| sort -nr

Enter the following data:

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 1c:4 b

:d6 :69: cd :07) ' -T fields -e frame. time|awk '{ print $4}'| head -1

$ tshark -r wlan . pcap -R '(( wlan .fc. type\_subtype == 0x20) && ( wlan .fc. protected == 1)) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 1c:4 b

:d6 :69: cd :07) ' -T fields -e frame. time|awk '{ print $4}'| tail -1

1. What was the span of time?
   1. Capture your screenshots.

The result seems unusual, and worthy of further inquiry. Typically, being unable to explain such a large proportion of network activity would be a huge red ﬂag. In this case, however, our sample size is so temporally short that it may only seem anomalous on a small scale. Perhaps some benign (if annoying) process on this device causes it to do the same thing every 10 minutes, and we’ve caught only one such burst.

Another interesting pattern to examine might be the distribution of communication partners with 00:23:69:61:00:ce (the WAP’s STA interface). It would be useful to ﬁnd out what Layer 3+ services this device provides (perhaps from Joe, or through our own inspection).

Typically DHCP is provided, as would be ARP service for the associated devices. In addition, most WAPs provide remote administrative access through the application layer—most commonly via HTTP (or better, HTTPS).

We’ve seen that there are data frames coming from the WAP’s BSSID and STA MAC addresses. Let’s drill down further and inspect the data frames sent by the WAP. We look ﬁrst at the distribution of destinations by number of data frames coming from the WAP’s BSSID (same value as before):

$ tshark -r wlan . pcap -R '( wlan .fc. type == 2) && ( wlan . bssid == *xx:xx:xx:xx:xx:xx* ) && ( wlan .sa == 00:23:69:61:00: d0 ) ' -T fields -e wlan .da | sort| uniq -c| sort -nr

Notice that Joe’s station (00:11:22:33:44:55) received far more data frames from the WAP’s BSSID interface than any other station on the network.

1. Capture your screenshot.
   1. How many data frames did Joe’s station receive?

Let’s also look at the data frames coming from the WAP’s STA interface (00:23:69:61:00:ce) and the distribution of their destinations.

$ tshark -r wlan . pcap -R '( wlan .fc. type == 2) && ( wlan . bssid == *xx:xx:xx:xx:xx:xx* ) && ( wlan .sa == 00:23:69:61:00: ce) ' -T fields -e wlan .da | sort| uniq -c| sort -nr

1. Capture the screenshot.

As we might reasonably expect, most of the data frames from 00:23:69:61:00:ce (the WAP’s STA interface) were sent to Joe’s station. After all, he’s likely logged onto the device’s web-based administrative interface in hopes of troubleshooting the situation. Given that Joe is the network administrator, if anyone is interacting directly with the WAP above Layer 2, it should be him. However, as we saw in the command output above, a signiﬁcant percentage of frames from the 00:23:69:61:00:ce were sent to the unknown station de:ad:be:ef:13:37. 54 It’s certainly reasonable to wonder why de:ad:be:ef:13:37 appears to have had signiﬁcant interaction directly with the WAP.

Why not look at the distribution of destination MAC addresses for data frames originating from de:ad:be:ef:13:37? That might give us some context:

$ tshark -r wlan . pcap -R '( wlan .fc. type == 2) && ( wlan . bssid == 00:23:69:61:00: d0 ) && ( wlan.sa == de: ad: be: ef :13:37) ' -T fields -e wlan .da | sort| uniq -c| sort -nr

It’s clear from the above output that within the time frame of this packet capture, most of the data frames sent from de:ad:be:ef:13:37 were directed to one other station—the WAP’s STA interface. Given that we don’t yet know who or what de:ad:be:ef:13:37 is, other than that it appears to have successfully associated with our WAP, it might be a good idea to place those frames in context. In particular, how does the traﬃc from de:ad:be:ef:13:37 ﬁt into our timeline? Let’s use tshark to obtain the start and end times of traﬃc originating from de:ad:be:ef:13:37.

$ tshark -r wlan . pcap -R '( wlan .fc. type == 2) && ( wlan . bssid == xx:xx:xx:xx:xx:xx) && ( wlan .sa == de: ad: be: ef :13:37) && ( wlan .da == 00:23:69:61:00: ce) ' -T fields -e frame . time|awk '{ print $4}'| head -1

$ tshark -r wlan . pcap -R '( wlan .fc. type == 2) && ( wlan . bssid == xx:xx:xx:xx:xx:xx) && ( wlan .sa == de: ad: be: ef :13:37) && ( wlan .da == 00:23:69:61:00: ce) ' -T fields -e frame . time|awk '{ print $4}'| tail -1

1. Where in the timeline did this traffic occur (e.g., early/mid/late)

**Management Frames**

Let’s start by obtaining a breakdown of the number of management frames by sender, to see if anything stands out. We can use tshark to show us this information:

$ tshark -r wlan . pcap -R '( wlan .fc. type == 0) && ( wlan . bssid == xx:xx:xx:xx:xx:xx) ' -T fields -e wlan .sa| sort| uniq -c| sort -nr

14858 00:23:69:61:00: d0

146 1c:4 b: d6 :69: cd :07

100 00:11:22:33:44:55

6 de: ad: be: ef :13:37

It is reasonable to expect the WAP to send out more management frames than the stations, but two orders of magnitude more seems unusual, especially when there are only a few stations.

Let’s examine the WAP’s outbound management frames (from the BSSID interface) and sort them by destination MAC address:

$ tshark -r wlan . pcap -R '( wlan .fc. type == 0) && ( wlan . bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 00:23:69:61:00: d0 ) ' -T fields -e wlan .da | sort| uniq -c| sort -nr

12217 1c:4 b: d6 :69: cd :07

2455 ff: ff: ff: ff: ff: ff

126 00:11:22:33:44:55

60 de: ad: be: ef :13:37

The overwhelming majority of management frames sent by the WAP’s BSSID interface were destined to one of the unknown stations—two orders of magnitude more than were sent to Joe’s station in the same time period. For that matter, the WAP’s BSSID interface sent more than 20 times as many management frames to the broadcast address as it did to Joe’s station.

Let’s view statistics regarding the subtypes of this traﬃc. In the output below, the ﬁrst column is the number of matching frames, the second column is the management frame subtype, and the third column is the destination MAC address.

$ tshark -r wlan . pcap -R '( wlan .fc. type == 0) && ( wlan . bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 00:23:69:61:00: d0 ) ' -T fields -e wlan .fc. subtype -e wlan .da| sort| uniq -c| sort -nr

12076 10 1c:4 b: d6 :69: cd :07

2454 12 ff: ff: ff: ff: ff: ff

118 5 00:11:22:33:44:55

Based on our analysis, the overwhelming majority of management frames from the WAP’s BSSID interface were sent to the unknown station 1c:4b:d6:69:cd:07, and were subtype 10 (0x0a): Disassociation. In other words, Joe’s WAP spent most of its frames trying to tell this station to “get lost.”

Its next-highest count of frames were broadcasting subtype 12 (0x0c): Deauthentication! In other words, it looks like Joe’s WAP also spent a bunch of time telling everyone to get lost! No wonder Joe was having problems.

Let’s look at the timing of these particular management frames. We’ll start by testing out a ﬁlter that should match all of the subtype 10 traﬃc within the WLAN of interest that was sent from the WAP’s BSSID interface to the unknown station 1c:4b:d6:69:cd:07. We should get 12,076 frames—the same as the number above:

$ tshark -r wlan . pcap -R '( wlan .fc. type\_subtype == 0x0a) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 00:23:69:61:00: d0 ) && ( wlan .da == 1c:4 b: d6 :69: cd :07) ' -T fields -e frame. time|wc -l

12076

That worked, so now let’s use the same ﬁlter with the “head” and “tail” programs in order to ﬁnd the time boundaries:

$ tshark -r wlan . pcap -R '( wlan .fc. type\_subtype == 0x0a) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 00:23:69:61:00: d0 ) && ( wlan .da == 1c:4 b: d6 :69: cd :07) ' -T fields -e frame. time|awk '{ print $4}'| head -1

08:59:42.221489000

$ tshark -r wlan . pcap -R '( wlan .fc. type\_subtype == 0x0a) && ( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 00:23:69:61:00: d0 ) && ( wlan .da == 1c:4 b: d6 :69: cd :07) ' -T fields -e frame. time|awk '{ print $4}'| tail -1

9:00:47.611120000

It appears that Joe’s WAP told 1c:4b:d6:69:cd:07 to Disassociate 12,076 times in the 65 seconds that passed from 08:59:42 to 9:00:47. That seems unusual.

While we’re at it, let’s look at the timing of the subtype 12 frames that the WAP was broadcasting (that second-most common type of management frame seen):

$ tshark -r wlan . pcap -R '( wlan .fc. type\_subtype == 0x0c) && ( wlan.bssid == xx:xx:xx:xx:xx:xx) && ( wlan .sa == 00:23:69:61:00: d0 ) && ( wlan .da == ff: ff: ff: ff: ff: ff)' -T fields -e frame. time|wc -l

2454

$ tshark -r wlan . pcap -R '( wlan .fc. type\_subtype == 0x0c) && ( wlan.bssid == xx:xx:xx:xx:xx:xx) && ( wlan .sa == 00:23:69:61:00: d0 ) && ( wlan .da == ff: ff: ff: ff: ff: ff)' -T fields -e frame. time|awk '{ print $4}'| head -1

08:59:03.241923000

$ tshark -r wlan . pcap -R '( wlan .fc. type\_subtype == 0x0c) && ( wlan.bssid == xx:xx:xx:xx:xx:xx ) && ( wlan .sa == 00:23:69:61:00: d0 ) && ( wlan .da == ff: ff: ff: ff: ff: ff)' -T fields -e frame. time|awk '{ print $4}'| tail -1

9:00:57.672520000

Joe’s WAP broadcast 2,454 Deauthentication messages during roughly the same time period (08:59:03 to 9:00:57). That also seems unusual.

Recall that the 802.11 speciﬁcation does not include a mechanism for verifying the authenticity of a sender. As a result, management frames can be spoofed. This leaves WLANs vulnerable to trivial denial-of-service attacks. Attackers can broadcast Disassociation or Deauthentication frames in order to cause network-wide outages or knock speciﬁc stations oﬀ the network. It is entirely possible that some of the Disassociation and Deauthentication messages shown above were actually sent by an attacker masquerading as Joe’s WAP, and not the WAP itself.

Could the traﬃc we’ve seen have caused problems for Joe? Of course. Joe’s station likely interpreted the Deauthentication messages as valid communications from the WAP, which would cause his station to attempt a new authentication/association negotiation. It is also possible that another station in physical range could have spoofed some or all of these messages.

Let’s investigate further.

Based on what we’ve seen so far, let us hypothesize that 1c:4b:d6:69:cd:07 is a suspicious actor (and for the moment our most obvious one, based on voluminous aberrant behavior).

Let’s catalog the activities of interest. We should make sure we record all the activities we see coming from this station and update our timeline.

The command below produces a count of frames sent by 1c:4b:d6:69:cd:07 sorted and counted by type and subtype. In the output below, the ﬁrst column is the number of matching frames, the second column is the 802.11 frame type, and the third column is the frame subtype.

$ tshark -r wlan . pcap -R '( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa == 1c

:4 b: d6 :69: cd :07) ' -T fields -e wlan .fc. type -e wlan.fc. subtype| sort -n| uniq -c| sort -nr

42816 2 0

77 0 11

69 0 0

As you can see, there were 42,816 data frames from 1c:4b:d6:69:cd:07, 77 Authentication Requests, and 69 Association Requests. Let’s sort by type/subtype to help us build our timeline further:

• The following command produces the timestamp for the ﬁrst Association Request (type 0 subtype 0) seen from 1c:4b:d6:69:cd:07:

$ tshark -r wlan . pcap -R '( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa

== 1c:4 b: d6 :69: cd :07) && ( wlan .fc. type\_subtype == 0x00) ' -T fields -e frame. time| head -1

• The following command produces the timestamp for the last Association Request (type 0 subtype 0) seen from 1c:4b:d6:69:cd:07:

$ tshark -r wlan . pcap -R '( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa

== 1c:4 b: d6 :69: cd :07) && ( wlan .fc. type\_subtype == 0x00) ' -T fields -e frame. time| tail -1

1. Capture your screenshots.
   1. Approximately how much time elapsed between the first and last packets?

• The following command produces the timestamp for the ﬁrst Authentication Request (type 0 subtype 11) seen from 1c:4b:d6:69:cd:07:

$ tshark -r wlan . pcap -R '( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa

== 1c:4 b: d6 :69: cd :07) && ( wlan .fc. type\_subtype == 0x0b) ' -T fields -e frame. time| head -1

• The following command produces the timestamp for the last Authentication Request (type 0 subtype 11) seen from 1c:4b:d6:69:cd:07:

$ tshark -r wlan . pcap -R '( wlan. bssid == 00:23:69:61:00: d0 ) && ( wlan .sa

== 1c:4 b: d6 :69: cd :07) && ( wlan .fc. type\_subtype == 0x0b) ' -T fields -e frame. time| tail -1

1. Capture your screenshots.
   1. Approximately how much time elapsed between the first and last packets?

**The Timeline**

Putting together the information we’ve gathered so far, we can start to build a timeline, as shown below (times are rounded to the nearest second):

* + - * + 08:56:41—Packet capture begins
        + 08:58:52 - Station “1c:4b:d6:69:cd:07” begins sending both Authentication and As- sociation Requests, essentially simultaneously, at a rate exceeding one per second
        + 08:59:03 - The WAP appears to begin broadcasting a larger ﬂood of Deauthentication messages
        + 08:59:42 to 9:00:47— The WAP appears to send 12,076 Disassociation messages to 1c:4b:d6:69:cd:07
        + 09:00:47 - The station “1c:4b:d6:69:cd:07” stops sending Authentication and Associ- ation Requests
        + 09:00:51 - The station “1c:4b:d6:69:cd:07” stops sending broadcasted data frames
        + 09:00:58 - The WAP’s apparent Deauthentication broadcasts stop
        + 9:02:14 to 10:03:33—de:ad:be:ef:13:37 sent small volume of data frames to the WAP’s STA interface (00:23:69:61:00:ce)
        + 9:03:35—Packet capture ends

**Final Questions:**

1. Based on the evidence, what is your theory?
   1. MAC address of unknown station
2. What are the BSSID and SSID of the WAP of interest?
3. Is the WAP of interest using encryption?
4. What stations are interacting with the WAP and/or other stations on the WLAN?
5. Are there patterns of activity that seem anomalous?
   1. Can you identify any of these patterns?
6. Can you identify any potential bad actors? (e.g. which station)

Can we determine if a bad actor successfully executed an attack? Let’s in- vestigate whether the WEP-cracking attack could have been successful. One way to ﬁgure that out would be to simply use aircrack-ng to try to recover the WEP key from the packet capture. If enough key material was revealed during that packet capture, we should be able to crack the key. Let’s ﬁnd out using an application that is not loaded into our system:

$ aircrack -ng -b 00:23:69:61:00: d0 wlan . pcap

Aircrack -ng 1.0

[00:00:02] Tested 938 keys (got 26805 IVs) KB depth byte( vote)

0 3/ 4 D0 (33536) 1F (33024) 27(33024) BC (33024) 2F (31744) 7B

(31744)

1 0/ 1 E5 (38656) 82(33024) 0C (32256) 3C (32000) EB (31744)

42(31488)

2 0/ 6 9E (34048) 27(33792) 7A (32768) E9 (32512) 8B (31744) 0E

(31744)

3 0/ 4 B9 (35328) D4 (35072) 2E (34048) B9 (33024) 00(32768)

06(32512)

4 8/ 10 6D (31488) 10(31232) B9 (31232) 7A (30976) 95(30976) A5

(30976)

KEY FOUND ! [ D0 : E5 :9 E: B9 :04 ]

Decrypted correctly: 100%

We were able to successfully recover the WEP key from the captured packets, which tells us that yes, it is certainly possible that the attacker’s WEP-cracking attack was successful. If we recovered the WEP key from these packets, then the attacker could have, too.

Recall that shortly after the apparent attack ended, the unknown station “de:ad:be:ef:13:37” successfully authenticated to the network, and there was addi- tional WEP-encrypted communication between “de:ad:be:ef:13:37” and Joe’s WAP. It is entirely possible that after recovering the WEP key with station 1c:4b:d6:69:cd:07, the attacker used the key to authenticate with station “de:ad:be:ef:13:37.” Now that we have recovered the WEP key, we can decrypt the Layer 2 traﬃc and investigate further using higher-layer protocol analysis techniques.

**Don’t forget to use the template and fill in all the sections.**

1. Network Forensics – Tracking Hackers through Cyberspace, by Sherri Davidoff and Jonathan Ham, Prentice Hall [↑](#footnote-ref-1)