Out Of Sight, But Not Out Of Mind: Traces Of Nearby Devices' Wireless Transmissions In Volatile Memory

By

Wicher Minnaard

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2014 EU

Amsterdam, NL (May 7th - 9th)

Digital Investigation 11 (2014) S104–S111

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Out of sight, but not out of mind: Traces of nearby devices' wireless transmissions in volatile memory

Wicher Minnaard

Netherlands Forensic Institute, Department of Digital Technology & Biometry, PO Box 24044, 2490 AA The Hague, The Netherlands

Keywords:
Network forensics
Wifi probe requests
Wifi beacons
IEEE 802.11
Network carving
Memory analysis

ABSTRACT

An IEEE 802.11 wireless device can leave traces of its presence in the volatile memories of nearby wireless devices. While the devices need to be in radio range of each other for this to happen, they do not need to be connected to the same network—or to any network at all. Traces appear in the form of full wire-type frames; a residue of the signals in the ether. We examine types of information that can be extracted from such residual frames and explore the conditions under which traces develop and persist. Their availability is determined by factors in both in the external environment (the types of signals in the ether) and the internal environment (the configuration and particulars of a device's wifi stack). To isolate some of these factors, we have created memory dumps of devices in various environments and configurations. Analysis of the dumps has offered insights into the conditions determining creation and decay of the traces. The results indicate that they will be available in a limited number of real-world scenarios. We conclude with practical advice on triaging and preservation.

Introduction

Since it was such a simple experiment that sparked our interest on the topic of residual wifi traces, we can best introduce the subject of this investigation by describing that initial experiment here.

A wifi-equipped[1] smartphone was brought in range of a wifi access point of which the RAM could be easily dumped. The smartphone had previously been connected to some networks, but it had never been connected to this particular access point. The phone was merely a passerby, as it were. On dumping the access point's memory, we were fascinated to find that, despite having no other relation to each other than simply having been in vicinity, names of networks to which the phone had connected previously could be found in the memory dump. Closer inspection of the dump showed that these names were embedded in the wire format of probe request frames; a format defined in the IEEE 802.11 standard. Thus it appeared that the frames had ended up *verbatim*, in network order, in the physical memory of the access point.

The implications of this find are quite unexpected for the smartphone user, who would certainly wonder why the name of his home network can be found in the memory of an access point which he simply happened to pass on his way to work. From an engineering perspective the phenomenon is less surprising—a data link frame may have to be buffered somewhere before can be decided that it should be discarded or unlinked. For some researchers in digital forensics, the phenomenon might not come as a complete surprise either. In 2011, Beverly, Garfinkel and Cardwell have addressed a related phenomenon (Beverly et al., 2011). Their work involves recovering "long-terminated network data in memory" of

E-mail addresses: wicher@holmes.nl, wicher@nontrivialpursuit.org.

[1] When using the terms '802.11' and the informal 'wifi', we are referring to radio technology based on the IEEE 802.11 standard (IEEE, 2007).

http://dx.doi.org/10.1016/j.diin.2014.03.013

which they find copious amounts—counter to their (and our) intuitions, which were *"[…] that portions of the Ethernet and/or IP protocol would be handled in hardware for performance reasons and not exposed to the operating system"*. As their goal is to extract IP and MAC addresses, they go on to develop a carver-generating technique that exploits statistical properties of the patterns in the context that the addresses appear in. This yields excellent results when compared to orthodox memory analysis; their efforts, as ours, show that there is information to be found beyond what is reachable by traversing the OS structures that reflect some 'current' state of the memory snapshot.

From a more general forensic perspective, the phenomenon of the retained probe request frames is interesting for the high potential value of this type of trace—precisely because the radio transmissions that lead to its creation are unintentional, ubiquitous, and unrelated to the nature of a crime. Any crime scene where wifi equipment can be found may thus contain these digital leads as to who was there, regardless of whether the crime itself had any digital component to it. In that respect the trace is rather like a digital version of a personal scent that lingers.[2]

The goals of our research into this novel type of trace are twofold: to identify types of information that can be discovered using recovered wifi frames, and to determine whether there could be real-world circumstances in which these wifi frames could be available for recovery. The latter is hardly a given—from a high level functional perspective there is no reason to keep them around for long (or at all, depending on frame contents).

Forensic artefacts in 802.11 management frames

In investigative law enforcement work, the goal is often to identify a person ('who'), to place him or her at a specific location ('where'), and preferably at a specific time ('when'). We show how these three goals relate to information present in two particular 802.11 management frame types: the beacon frames transmitted by the access point (AP), and the broadcast probe request frames transmitted by the wifi device (STA).

Preliminaries

To be able to discuss 802.11, we need to establish some common ground. In this paper we will refer to, but not reiterate frame formats defined in the 2007 1184-page IEEE 802.11 specification.[3] To aid in quickly understanding the gist of the parts of the standard relevant to our research, we offer some comments on sections in the standard that we deem particularly informative.

§**5.2.3 Distribution system (DS) concepts**

Central to 802.11 is the notion of a 'base service set' (BSS), which, in an infrastructure network, is a set of stations (STAs) served by an access point (AP). Multiple BSSes can be connected to form an extended service set (ESS), but in the common domestic case the ESS consists of just one BSS, one AP, and some STAs. Ultimately the SSID identifies an ESS and is what the end user sees as "the network name". The SSID is not used for addressing purposes (the ESS is not logically addressable; an SSID is an identifier rather than an address) and is present in only a limited number of packets.

§**7.1.3.3 Address fields**

Frames can contain up to four addresses. This section explains their meaning. The base service set is addressable through the BSSID. In the common 'infrastructure' network, this address is equal to the MAC address of the wireless interface of the AP. In an 'ad-hoc' network without APs—an IBSS, not to be confused with the 'infrastructure' BSS—this is not the case.

§**11.1.2 Maintaining synchronization**

This section introduces the 'Timing Synchronization Function', which is at the basis of the timestamp field that we will encounter in beacon frames. The TSF is a timer with μs resolution, and its value is stored in a 64-bit wide field. Interestingly, this results in a surprisingly large rollover time of almost 585,000 years. In an IBSS it is used in a mechanism to orchestrate the generation of beacon frames across the IBSS. The timer previously also played a role in the coordination of a frequency hopping mechanism that was enabled in the original 1997 802.11 standard.

§**11.1.3 Acquiring synchronization, scanning**

This section explains how a station can find what users call 'networks'—ESSes and their BSSes. It describes the passive approach in which a station quietly listens for beacon frames as well as the active approach of sending out probe request frames. It does not describe why a station would prefer one or the other, but it raises a question: Why do we have probe request frames when we already have beacon frames for which we could simply listen?

Obviously, a probe request is needed to find a certain 'hidden' network that does not broadcast its SSID in its beacons. Less obvious is the fact that an active scan may be preferred since it can save power—it finishes much sooner since probe responses typically come in within tens of milliseconds, whereas beacon intervals and thus the required channel dwell times can be in the order of seconds. Since the antenna circuit can be powered down when further traffic is not expected, energy savings can be significant. A second benefit of scanning actively is that by receiving a probe response it is not only established that the device can hear the AP—it is also established that the AP can hear the device.

§**11.2 Power management**

This section deals with power management. A STA can signal the AP to buffer packets destined for it while it sleeps for a moment. This implies that even when a STA is associated, it is not necessarily continuously listening.

[2] Testament to the forensic utility of such smartphone scent trails is the fact that they are being exploited commercially, most infamously with the recent deployment of 'smart' litter bins on the streets of the City of London (BBC News, 2013).

[3] The 2007 specification is available through IEEE Get, http://standards.ieee.org/getieee802/download/802.11-2007.pdf. The 2012 specification is over twice its size and its additions are not relevant to our research.

To preserve readability we will avoid excessive use of acronyms in the remainder of this paper, preferring colloquial terms over formal terms. And unless noted otherwise, we will remain in the context of simple infrastructure mode networks, consisting of just one basic service set; one access point and zero or more associated stations.

The beacon frame

Beacon frames are sent by access points to advertise the name and characteristics of the service set that they are managing. Let us assume that we have found some beacon frame in the memory of a wifi-equipped device. From inspecting its fields, we will be able to learn the *where* and the *when.*

Beacons: location of transmission

We know the identity of the access point that has sent the beacon through the BSSID (MAC address) and SSID (network name) fields. To have received the beacon frame, the device must have been in range of the access point. If the access point is part of a crime scene we obviously already know its location. If we then find one of the access point's beacon frames in the memory of the device, it is highly likely that the device has been near the access point. But what if the location of the access point where the beacon frame originated is unknown to us? To find where the device has been, we need to relate the access point's identity to a geographical location. As such is a common need in the context of smartphone geolocation, we are not without options: there are a number of geolocation databases[4] which yield coordinates given BSSIDs or SSIDs. However, many freely accessible databases will only answer queries when given two or more identities which are in close proximity of each other. This is to thwart attempts at following particular access points around; it protects some of the privacy of access point owners. However, the Wigle[5] database is one that can be freely queried using just one single BSSID or SSID.

Apple's database can also be queried using a single BSSID. Apple does not offer a public API, but a description of the protocol used to access it is available.[6]

Beacons: time of transmission

The beacon frame features a timestamp ts_1 that corresponds with the state of the access point's timer at a time t at which the device was near the access point. We know that the timer runs at 1 MHz, but we do not know when this timer was started. To derive t we will therefore need to capture a second beacon from the same access point, containing a second timestamp ts_2. If we note the time t' (in microsecond since some epoch) at which we have captured the second beacon, we can derive t using the following formula:

$$t = t' - (ts_2 - ts_1)$$

But what if the timer has been reset between t and t'? In that case ts_2 represents a different timer run than the one in which we are interested. These timer resets are to some extent detectable. The trivial case is when $ts_2 < ts_1$; since we know that we have captured ts_2 at a later point in time than ts_1, the former should be larger than the latter. It is harder to detect the situation in which the timer has been reset, but enough time has passed for ts_2 to surpass ts_1. If ts_1 is a low value this is a real possibility.

The probe request frame

Probe request frames are sent when a wifi device actively scans for networks. Since the identifier of a network is the SSID and not the BSSID, the frames are addressed to the broadcast BSSID. They may be used to probe for the wildcard SSID (a zero length SSID), but may also be used to probe for a particular network. In that case the frame will contain the name of that network. Consumer devices commonly keep a list of networks that they have previously connected to, and will regularly scan for these networks.

Suppose we have found some probe request frames in the memory of a wifi-equipped device. What information can we extract from them?

Probe requests: device identity

The identity of the device is revealed through the sender address field. This is the MAC address of the wireless device. The first three octets of the MAC address form the vendor prefix. Vendor prefixes are centrally administered, and many of them are publicly listed.[7] Using the prefix we can to some extent determine the brand of the device. When probe requests are recovered at a given site, a person carrying a wifi device may be linked to that site based on the identity of his or her device. Even if the device itself and its carrier are unavailable, extracted information may be of tactical use if the device identity appears at multiple sites. In the context of crime scenes, this is useful in order to establish that seemingly isolated incidents are indeed related after all.

Probe requests: previous networks

A name of a network to which device has previously connected to can be revealed through the SSID field. Coupled with the aforementioned geolocation databases this can offer insights into the whereabouts of the device, especially if multiple probe request frames, yielding multiple network names, can be recovered. Overlapping sets of network names may also be used to establish relations between multiple device identities—but this needs to be done with consideration for the fact that some network names are too common to indicate relationships. The authors of *Uncovering Social Relationships through Smartphone Probes* (Barbera et al., 2013) offer a technique to adjust relationship strength for popularity of network names.

[4] Mozilla offers a list of free and commercial databases at https://wiki.mozilla.org/Services/Location/Bootstrap.

[5] See http://wigle.net.

[6] The iSniff project contains a protocol implementation, see https://github.com/hubert3/iSniff-GPS.

[7] See http://standards.ieee.org/develop/regauth/oui/public.html.

Recovering frames by searching for signatures

Both frame types have readily recognizable features. This makes it particularly easy to search for them in a raw memory dump using nothing more than regular expressions. Beacon frames start with 0x80 00 00 00, followed by the broadcast address (6 × 0xFF), the transmitter MAC address and the BSSID. Since the latter two are identical in infrastructure networks, a regular expression in Python syntax can be formulated as:

```
b'\x80\x00\x00\x00(\xff){6}(.{6})\\2'
```

Probe request frames feature similar regularities: they start with 0x40, then 0x00 or 0x10 depending on whether power saving features are enabled, then 0x00 00, the broadcast address, the transmitter MAC address, and the BSS broadcast address (6 × 0xFF). As a Python regular expression, this becomes:

```
b'\x40(\x00|\x10)\x00\x00(\xff){6}(.{6})\\2'
```

The receive path: driver-firmware interaction

Through what mechanism do the frames enter main memory? To derive a general model of operation we examine the 'iwlwifi' driver source code from the Linux (3.12.2) kernel. The drivers/net/wireless/iwlwifi/pcie/rx.c file contains comments that explain the mechanism. Keep in mind that the interface card operates to some extent independently from the host machine. It has its own processor and features a separate environment in which it executes its firmware.

The way in which the driver (which lives in kernel space) and the firmware (running on the device processor) exchange messages—packets, commands, command results—is by placing them in regions of host physical memory accessible by both. The driver allocates kernel memory for a circular bookkeeping buffer, sets up DMA for that memory region, and hands the bus addresses for that region to the device firmware. The firmware knows how to work with these bookkeeping entries; it is part of the driver-firmware interface for this particular family of devices. Each slot in the bookkeeping buffer describes a message buffer—these, in turn, are allocated by the driver in host memory, and are DMA-enabled. Two indices (again, in shared memory) to entries in the bookkeeping buffer are kept. The 'READ' index is managed by the firmware and points to the first slot from which the firmware doesn't want the driver to read (the firmware may be writing to the associated memory). The 'WRITE' index is managed by the driver and points to the last slot from which the driver has read—subsequent slots should not yet be reused by the firmware.

The firmware can now place incoming packets in host RAM at the addresses described by the slots in the bookkeeping buffer, taking care not to use slots from which the driver hasn't read yet. It updates the READ index and fires off an interrupt request. At that point the driver wakes up and queries the READ index to check which slots to process. While processing packets the driver updates the read slots with fresh, possibly newly-allocated memory buffer addresses and advances the WRITE index.

The layer of indirection allows the driver to buffer more packets than can be described in the bookkeeping structure. It also enables zero-copy mechanisms; received packets are handed to higher parts of the kernel's network stack by passing the pointer, not the packet. The implication of this is that the packets that we are so interested in may be located in any physical memory location for which DMA transfers could have been arranged—and the driver may hold no reference to them anymore. On the other hand, the iwlwifi driver recycles any message buffers that do not need to be passed on. Where the frames and the references to them eventually end up is therefore very much dependent on runtime circumstances. So while the bottom-up static analysis of this specific driver helps in understanding *why* we will sometimes find residual wifi packets in RAM, it does not allow us to make general predictions on *when* we can expect them. Therefore, we will take an empirical approach to find out under which circumstances traces can develop and persist.

Factors determining retention and decay

Experimentation method

A handful of wireless devices were configured in various modes and submitted, one at a time, to finely controlled radio traffic: probe request frames, beacon frames, and data frames. After or during exposure, the device's RAM was dumped and analyzed for traces of the frames emitted earlier.[8]

The design of the experiments was as follows: A transmitter and the subject device were placed inside a Faraday cage.[9] In some experiments they were joined by an access point for the subject device to associate with. The experiments were automated through a control unit, which instrumented the transmitter and the subject device through an optical wired network connection; all manners of frame injection, configuration, and dumping memory contents were scripted.

The transmitted probes and beacons were generated so as to be unique across experiments. In each frame the 32-byte SSID and the 6-byte sender MAC address are derived from the MD5 hash of a sequence number salted with the experiment's name. Because of this, each transmitted frame later retrieved from a dump could be uniquely identified; the details of each transmitted frame were logged. These logs also allowed the searching of memory dumps for duplicates of the SSID and the transmitter MAC.

Table 1 lists details of radio devices used. Table 2 lists the software configurations in which they are used. From these tables it follows that the combination of OS kernel (Linux or

[8] The scripts used to inject and extract frames are made publicly available, see http://oos.nontrivialpursuit.org/ogb/framescripts.tar.gz.

[9] See http://www.forensicinstitute.nl/products_and_services/forensic_products/faraday_cage/.

Table 1
Wireless chipsets used.

Chipset	Interface	Host system	Description
bcm4330	SDIO	Samsung Galaxy Nexus	A Broadcom chip on an ARM smartphone board, brcmfmac driver.
bcm4321	PCI	D-link DSL-2740B	On a Broadcom MIPS wifi router board (bcm6358), b43 driver.
ar9103	AMBA	TP-Link TL-WR1043ND	On an Atheros MIPS wifi router board (ar9132), ath9k driver.
ar9485	PCIe	HP Pavilion DM1-4000SD	An Atheros adapter, part of the laptop. Linux driver: ath9k, Windows manufacturer driver version: “9.2.0.427”.
rt2070	USB	HP Pavilion DM1-4000SD	A Ralink chip inside a D-link DWL-G122 (rev.E1) USB adapter. Linux driver: rt2800usb, Windows manufacturer driver version: “ v3_60_s0038”.

Table 2
Host system configuration.

Host system	Description
Galaxy Nexus	Equipped with the Replicant 4.0-0003 fully open source Android-compatible firmware (Linux kernel with Android userspace). Memory is dumped with LiME r17 (Sylve et al., 2012), using TCP over ADB. This system has 768 MB of RAM.
DSL-2740B TL-WR1043ND	Running OpenWRT 12.09—a Linux distribution targeted at networking equipment and embedded systems. Memory is dumped over TCP using LiME r17. Both systems have 32 MB of RAM.
Pavilion DM1-4000SD	Windows 7 Home Premium, 32-bit, build 7601 (SP1). Memory is limited to 512 MB and its contents are acquired using crash dumps. For the decay tests memory is dumped over TCP using KnTDD 2.3.0.2733. 32-bit Linux through the SystemRescueCD live-cd distribution with kernel 3.4.52, limited to 256 MB RAM, which is dumped over TCP using LiME r17.

Windows) and the wireless chipset suffice to identify a configuration.

Results

Retention

In this experiment, 1100 frames were injected in 11 seconds, and device memory was dumped immediately afterwards. Dumps were then searched for any trace of the 1100 aired frames. For each combination of device and mode, the experiment was repeated 10 times (each time using different frames—note that all frames are unique).[10] The results are shown in Fig. 1.

Before comparing measurements it is important to recognize the following:

1. The time needed to perform the dump varies with memory size and method, and time could be a factor in retention.
2. On Windows, we were not able to pin wifi interfaces to any particular channel if the interface was not configured to connect to some network. In all Windows experiments except the ones in the “client mode, connected” state it is therefore undefined as to which frequency the subject device was tuned, while in the other cases we had the subject device tuned to the same channel on which we were sending our frames (2437 MHz).
3. It was not possible to avoid Windows sending probe requests of its own making. This has consequences for experiments with Windows in the “Connected” state, for in that setup the subject device is joined by an access point, which will respond to the probes with probe response frames.
4. In all “Connected” state experiments, the access point was sending beacon frames at a rate of 10 per second.

With these considerations in mind, we observe:

1. Both access point platforms retain probe request frames when in AP mode, as witnessed in the initial experiment described in the introduction to this paper.
2. The same does not hold when in disconnected client mode, in which the bcm4321 retains around 250 frames while the ar9103 retains none. This can also be seen in the case of Linux coupled with the rt2070 chip. In ad-hoc mode retention is much lower than in disconnected client mode.
3. Comparing the Windows and Linux drivers for the rt2070 we see that both retain frames in client mode. The Windows driver retains less frames than the Linux driver, and both show lower retention when connected to an access point.
4. The ar9485 shows hardly any retention at all.

Retention on bcm4330 (Android). Absent from Fig. 1 is the bcm4330 chip (on the Android smartphone). We were not able to detect any retention for that configuration, be it in AP or client mode (either connected or disconnected). Even when injecting on wifi channels 1 through 11 *while* dumping the memory, and disabling all power saving options,[11] we were not able to retrieve any frame, MAC address, or SSID. The only situation in which we were able to retrieve any trace of what we had sent was when injecting beacon frames while dumping, with the Android wifi network chooser application active—in which case we were (unsurprisingly) able to capture the SSIDs that were

[10] Samples of memory dumps have been made publicly available as http://oos.nontrivialpursuit.org/ogb/datasamples.tar. A complete archive of memory dumps is available on request.

[11] Android aggressively saves power, see https://developer.android.com/reference/android/net/wifi/WifiManager.html. The “WakeLock - PowerManager” app, available from https://play.google.com/store/apps/details?id=eu.thedarken.wl lets one disable various power saving options.

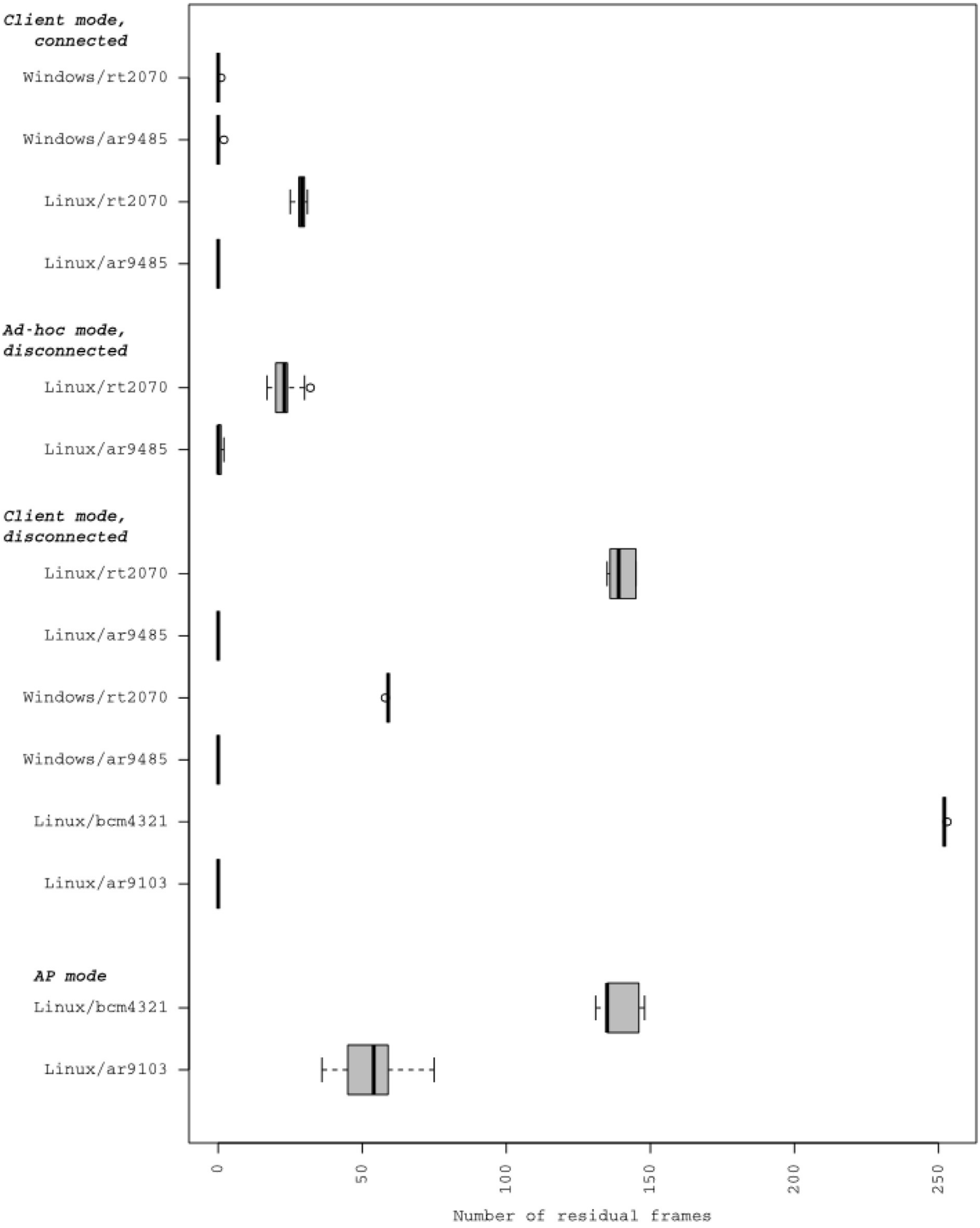

Fig. 1. Box-and-whisker plot of probe request frame retention with various OSes, chipsets and modes. $N = 10$. All chipsets show retention in one or more configurations, but there is significant divergence in the amount of retained frames and in the variance over multiple runs.

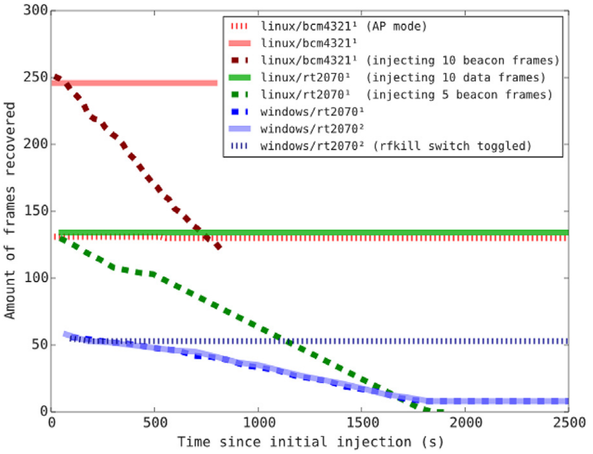

Fig. 2. Decay of frames over time for probe request frames (^{1}) and beacon frames (^{2}).

shown on screen. We suspect that the lack of residual frames is related to the device and driver architecture. On Linux, there are two classes of wireless device drivers: SoftMAC and FullMAC. They differ in the extent to which host software (the driver, in kernel space) is involved in managing the interface. SoftMAC drivers need to handle state management themselves, while FullMAC drivers leave this to the firmware.[12] As it happens, the driver for the bcm4330 (brcmfmac) is of the FullMAC class while all of the other drivers that we use (ath9k, b43, rt2800usb) are of the SoftMAC variety.

Decay

The observed difference in retention between the connected and disconnected states for rt2070 raises the question of whether it is possible that the beacon frames emanating from the access point are pushing out probe request frames. And more generally, what is the influence of radio traffic and time on retention?

To determine this, 1100 probe requests or beacons were injected once. Thereafter the memory of subject systems was dumped at regular intervals, and for each dump a count was made of the amount of the previously injected frames retrievable. In four of the experiments an additional parameter was tested. Three of these concerned the injection of beacon or data frames between the taking of memory snapshots, to simulate certain types of radio traffic. The beacon frames were addressed to the broadcast address, while the data frames had a sender and destination addresses unrelated to the subject system. A fourth external parameter was the RFKILL state. RFKILL is a mechanism to disable radio transmissions and can be implemented as a software switch, hardware switch, or both. Only the bcm4321 and rt2070 chips are used, as they show the greatest retention.

Fig. 2 graphs decay over time for different scenarios. Four observations can be made for these experiments:

1. Injection of beacon frames results in decay. This aligns with observation #3 from the retention experiments—less probe request frames could be retrieved when the subject device had been exposed to beacon frames sent by the access point with which it was associated.
2. Injection of traffic not addressed to the device (the data frames) does not result in decay.
3. The Linux devices, left to themselves, show no decay (flat lines). The Windows configuration shows decay for both probe request frames and beacon frames. Oddly, decay on the Windows system stops when just 8 frames are left.
4. Similarly peculiar, shutting down the radio by toggling the RFKILL switch prevents such decay on Windows.

Odds and ends

Traces in the Windows hibernation file. Residual frames can be recovered from volatile memory, but they may also end up on disk when the host OS goes into hibernation. We tested this for the Windows/rt2070 combination, using the Volatility framework to convert the hiberfil.sys hibernation file to a raw image. We were able to recover 5 frames. These frames were among the last 10 injected frames.

Amplification and wlanext.exe. On Windows, we witnessed several occurrences of amplification—some frames were duplicated and MAC addresses and SSIDs belonging to a single frame could be found at multiple memory locations.

Fig. 3 shows such amplification for a memory dump of Windows/rt2070. The top plot shows that of the 1100 beacon frames injected, it is not necessarily just the ones injected last that remain. Among the last frames are some that are duplicated. The middle subplot shows amplification of the access point's MAC address. Note that MAC address naturally occurs twice in a beacon frame; once as the transmitter MAC address and once as the

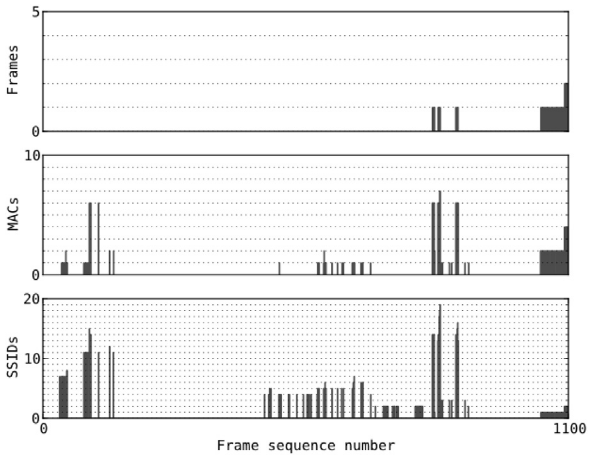

Fig. 3. Amplification of beacon frames and associated SSIDs and MAC addresses. The scale of the middle subplot is twice that of the one above and half that of the one below.

[12] See http://wireless.kernel.org/en/developers/Documentation/Glossary#SoftMAC and section 10.1 of IEEE 802.11.

BSSID—therefore, the rightmost cluster does not depict actual amplification. To the left of that cluster we do see some amplification, and we find many MAC addresses related to frames that are no longer available in full. The same phenomena show up when comparing the middle and bottom plot. There we see even more amplification—as much as 19×, and we find many SSIDs that belong to frames no longer available.

We have not been able to pin down the mechanism that underlies this amplification. We suspect it is related to the wlanext.exe system process. The description embedded in the executable is *Windows Wireless LAN 802.11 Extensibility Framework*. Analysis with Volatility[13] shows this process present in the process list of each of the dumps that feature amplification, while it is absent from the memory images which do not show any amplification.

Conclusions

Where do these findings leave us? We have shown the potential value of wireless management frames for forensic investigations. The results of exploring the residual frame phenomenon add to the prior work that investigates traces of low-level network structures. They also raise questions pertaining to the oddities observed, which point at a problem that surfaces when trying to generalize from these sparse results. They are certainly not sufficient to fully model retention and decay in terms of interactions between userspace, drivers, firmware, and the external environment. It is hard to imagine how many permutations of these elements one would need to test to arrive at a trustworthy model—this is the classic problem of induction. Still, we have become convinced that the occurrence of residual frames is not an uncommon phenomenon. It can occur in access points and client devices alike, even when they are not connected to any network. The information is quite volatile however, and quirky behavior can be expected as driver implementations are subject to the whim of their developers.

Turning to more practical matters, are the findings of use when assessing a crime scene? They are—to some extent. We have seen that broadcast traffic is destructive. Beacon frames are broadcast traffic, so that precludes the majority of urban locations. Probe request frames from other devices are also broadcast traffic, precluding busy locations. Thus it appears that it is only in remote areas—exactly in settings where potential witnesses are few and far between—that this type of trace could show its value, pointing to whoever was last at the crime scene. And as frames can be found OS- and device-agnostically using a simple regular expression, it is worthwhile searching for them when hibernation files or memory dumps can be made available. The latter is not a given; we cannot talk lightly about dumping volatile memory of arbitrary devices.

Once at a crime scene, there are measures that an investigator can take to preserve traces. He may be able to toggle an RFKILL hardware or software switch (reachable in Windows through the win+X key combination). Suspending and hibernating the device can also be considered, but these actions will usually lock a device.

We think a great challenge lies in recognizing the infrequent situations in which the trace type can be made use of. Furthermore, given the fragility of the traces, the first responder's awareness must be raised in order to prevent crime scene contamination by signals from wireless devices carried by these first responders.

Acknowledgments

This research project has been funded by NCTV [14] under research grant #395219. This funding source had no involvement in study design; nor in the collection, analysis and interpretation of data; nor in the writing of the report; nor in the decision to submit the article for publication.

I would like to thank all colleagues who have helped, and especially Susan Laraghy, for proofreading, and Marnix Kaart, for advice on research targets.

Furthermore, I thank the open source community for making available excellent tools and freely accessible technical information.

References

Barbera MV, Epasto A, Mei A, Perta VC, Stefa J. Signals from the crowd: uncovering social relationships through smartphone probes. In: Proceedings of the 2013 Conference on Internet Measurement Conference, IMC '13. New York, NY, USA: ACM; 2013. pp. 265–76. http://dx.doi.org/10.1145/2504730.2504742.

BBC News. City of London calls halt to smartphone tracking bins. URL, http://www.bbc.co.uk/news/technology-23665490; August 2013.

Beverly R, Garfinkel S, Cardwell G. Forensic carving of network packets and associated data structures. Digit Investig 2011;8(Suppl.):S78–89. http://dx.doi.org/10.1016/j.diin.2011.05.010.

IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks-Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-2007. http://dx.doi.org/10.1109/IEEESTD.2007.373646.

Sylve J, Case A, Marziale L, Richard GG. Acquisition and analysis of volatile memory from android devices. Digit Investig 2012;8(3):175–84. http://dx.doi.org/10.1016/j.diin.2011.10.003.

[13] Volatility, http://code.google.com/p/volatility/.

[14] National Coordinator for Security and Counterterrorism, http://english.nctv.nl/.