Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting[*]

Jason Franklin[1] Damon McCoy[2] Parisa Tabriz[3] Vicentiu Neagoe[4]
Jamie Van Randwyk[5] Douglas Sicker[6]

Abstract

Motivated by the proliferation of wireless-enabled devices and the suspect nature of device driver code, we develop a passive fingerprinting technique that identifies the wireless device driver running on an IEEE 802.11 compliant device. This technique is valuable to an attacker wishing to conduct reconnaissance against a potential target so that he may launch a driver-specific exploit.

In particular, we develop a unique fingerprinting technique that accurately and efficiently identifies the wireless driver without modification to or cooperation from a wireless device. We perform an evaluation of this fingerprinting technique that shows it both quickly and accurately fingerprints wireless device drivers in real world wireless network conditions. Finally, we discuss ways to prevent fingerprinting that will aid in improving the security of wireless communication for devices that employ 802.11 networking.

1 Introduction

Device drivers are a primary source of security holes in modern operating systems [1]. Drivers experience error rates of three to seven times higher than other kernel code, making them the poorest quality code in most kernels [2]. There are a large number of different device drivers available, each being a potentially large body of code that is frequently modified to support new hardware features. These factors and the fact that drivers are often developed by programmers who lack intimate knowledge of the operating system kernel contribute to the disproportionately high number of bugs found in device drivers [3].

In general, device drivers execute in kernel space; hence, exploiting a vulnerable driver leads to compromise of the entire operating system. This threat is somewhat tempered by the fact that interacting with a driver typically requires physical access to a system. As a result, most security holes in device drivers are difficult to exploit remotely. For instance, it is hard to remotely interact with, much less exploit, a video or keyboard driver. Classes of drivers exist with which it is possible to interact without physical access to a system. Drivers for network devices such as wireless cards, Ethernet cards, and modems are examples. In particular, wireless network device drivers are easy to interact with and potentially exploit if the attacker is within transmission range of the wireless device. Today, the single most common and widespread wireless devices are those conforming to the IEEE 802.11 standards [4]. The vast number of 802.11 devices, the ease with which one may interact with their drivers, and the suspect nature of driver code in general has led us to evaluate the ability of an attacker to launch a driver-specific exploit by first fingerprinting the device driver.

Fingerprinting is a process by which a device or the software it is running is identified by its externally observable characteristics. In this paper, we design, implement, and evaluate a technique for fingerprinting IEEE 802.11a/b/g wireless network drivers. Our approach is based on statistical analysis of the rate at which common 802.11 data link layer frames are transmitted by a wireless device. Since most wireless exploits are dependent on the specific driver being used, wireless device driver fingerprinting can aid an attacker in launching a driver-specific exploit against a victim whose device is running a vulnerable driver.

Our technique is completely passive, meaning that a fingerprinter (attacker) needs only to be able to monitor wireless traffic from the fingerprintee (target, victim). This makes it possible for anyone within transmission range of a wireless device to fingerprint the device's wireless driver. Passive fingerprinting techniques have the advantage over active approaches in that they do not transmit data, making prevention of such techniques dif-

[*]This paper was originally published in Proceedings of USENIX Security '06, July 31 - August 4, 2006, Vancouver, BC, Canada

ficult. If an attacker can passively determine which driver a device is using, he can successfully gain information about his victim without fear of detection.

Our fingerprinting technique relies on the fact that most stations actively scan for access points to connect to by periodically sending out probe request frames. The algorithm used to scan for access points is not explicitly defined in the 802.11 standard. Therefore, it is up to the developers of device drivers to implement their own method for probing. This lack of an explicit specification for a probing algorithm in the 802.11 standard has led to the development of many wireless device drivers that perform this function entirely differently than other wireless device drivers. Our fingerprinting technique takes advantage of these implementation-dependent differences to accurately fingerprint a driver. Specifically, our method is based on statistical analysis of the inter-frame timing of transmitted probe requests. A timing-based approach has a number of advantages over a content-based approach. Primary among these is the fact that coarse-grained timing information is preserved despite the encryption of frame content as specified by security standards such as Wired Equivalent Privacy (WEP) or 802.11i [5].

Fingerprinting an 802.11 network interface card (NIC) is not a new concept. Many tools exist, such as Ethereal [6], that use the wireless device's Media Access Control (MAC) address to identify the card manufacturer and model number. A MAC address is an ostensibly unique character string that identifies a specific physical network interface. The IEEE Standards Association assigns each NIC manufacturer a special three-byte code, referred to as an Organizationally Unique Identifier (OUI), which identifies a particular manufacturer. While not part of the standard, most manufacturers use the next byte to specify the model of the NIC. There are a few notable advantages to using our method instead of relying on the information contained in the captured MAC address. First, the MAC address only identifies the model and manufacturer of the NIC. Our technique fingerprints the device driver (which resides at the operating system level), where the bulk of exploits rest. Second, some NICs can operate using multiple drivers, implying that the MAC address would not be enough information to identify what driver the NIC was using. Finally, whereas the MAC address is easily alterable in most operating systems, the features used by our passive technique are not a configurable option in any of the drivers tested.

Our testing demonstrates an accuracy for our method in identifying the driver that ranges from 77-96%, depending on the network setting. Our technique requires only a few minutes worth of network data to achieve this high level of accuracy. We also confirm that the technique can withstand realistic network conditions.

Contributions The main contributions of this paper is the design, implementation, and evaluation of a passive wireless device driver fingerprinting technique. Our technique is capable of passively identifying the wireless driver used by 802.11 wireless devices without specialized equipment and in realistic network conditions. In addition, we demonstrate that our technique is accurate, practical, fast, and requires little data to execute.

The remainder of the paper is organized as follows. Background material is presented in Section 2. Section 3 presents the design for our wireless device fingerprinting technique. Section 4 describes the implementation of our fingerprinting technique and Section 5 presents our experimental results and evaluation of our technique under realistic network conditions. Section 6 presents the limitations of our technique and Section 7 discusses possible ways to prevent driver fingerprinting. Finally, Section 8 examines related work and we conclude in Section 9.

2 Background: IEEE 802.11 Networks

Wireless technologies are encroaching upon the traditional realm of "fixed" or "wired" networks. The most widely adopted wireless networking technology thus far has been the 802.11 networking protocol, which consists of six modulation techniques, the most of common of which are the 802.11a, 802.11b, and 802.11g standard amendments. The price erosion and popularity of 802.11 capable hardware (especially 802.11b/g) has made wireless networks both affordable and easy to deploy in a number of settings, such as offices, homes, and wireless hot spots. Because of this, 802.11 is currently the most popular and common non-telephony communication protocol available for wireless communication [7].

The 802.11 standard defines a set of protocol requirements for a wireless MAC, or medium access control, which specifies the behavior of data link layer communication between stations in a wireless network. A station is simply a device with wireless capabilities, such as a laptop or PDA with a wireless networking interface. Throughout this paper, we often refer to stations as clients. Most 802.11 networks operate in infrastructure mode (as opposed to ad-hoc mode) and use an access point (AP) to manage all wireless communications; it is this type of network that is the setting for our fingerprinting technique. An example of a simple infrastructure network with three clients and one access point is depicted in Figure 1.

A key component of the 802.11 standard is the MAC specification that outlines the function of various communication frames. The MAC coordinates access to the wireless medium between stations and controls transmission of user data into the air via control and management frames. Higher-level protocol data, such as data produced by an application, is carried in data frames.

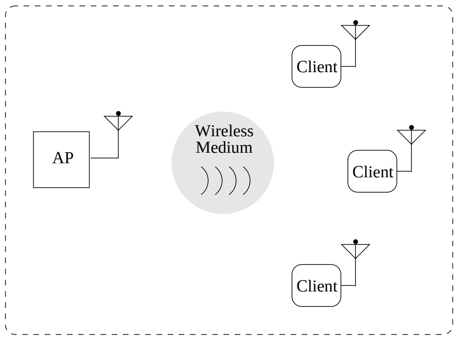


Figure 1: An infrastructure mode IEEE 802.11 network.

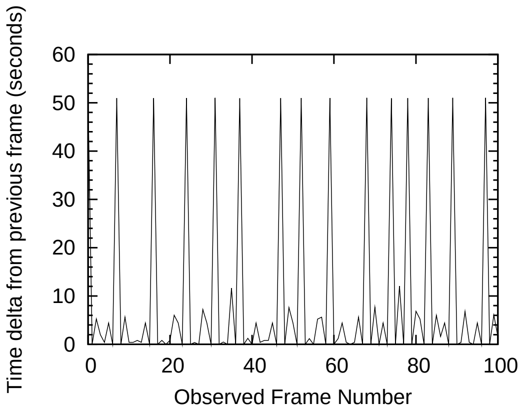
All 802.11 MAC frames include both a type and subtype field, which are used to distinguish between the three frame types (control, management, and data) and various subtypes. We consider only management frames in our passive fingerprinting technique, and specifically focus on probe request frames. Because of this, we only describe the most pertinent MAC frames communicated when a client joins a wireless network, and refer the reader to the IEEE 802.11 standard specification [4] for a more detailed description of MAC framing.

Each mobile client must identify and associate with an access point before it can receive network services. In a process called active scanning, clients use *probe request* frames to scan an area for a wireless access point, providing the data rates that the client can support inside fields of the probe request. If an access point is compatible with the client's data rates, it sends a probe response frame to acknowledge the request. Once a client identifies a network and authenticates to the access point via an authentication request and authentication response, the client can attempt to join the network by issuing an association request. If the association is successful, the access point will respond to the client with an association response that includes a unique association ID for future communications. At this point, all communication between a client and another machine, whether it resides within the wireless network or is located outside of it, is routed through and controlled by the access point.

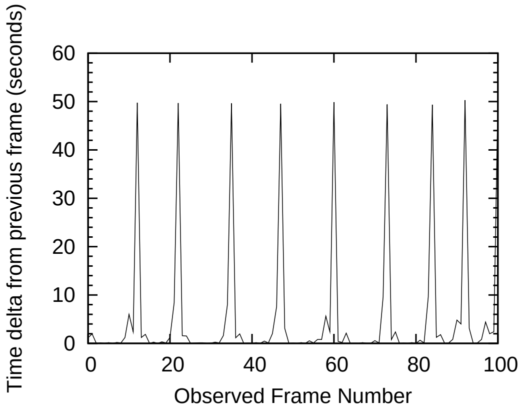
3 Fingerprinting Approach

Our fingerprinting technique is solely concerned with the active scan function in wireless clients. When actively scanning, clients send probe request frames to elicit responses from access points within transmission range. The IEEE 802.11 standard describes the active scan function of a client as follows. For each channel, the client broadcasts a probe request and starts a timer. If the timer reaches *MinChannelTime* and the channel is idle, the client scans the next channel. Otherwise, the client waits until the timer reaches *MaxChannelTime*, processes the received probe response frames and then scans the next channel. Further detailed specification of the active scanning function is not provided in the IEEE 802.11 standard. As a result, implementing active scanning within wireless drivers has become a poorly guided task. This has led to the development of many drivers that perform probing using slightly different techniques. By characterizing these implementation-dependent probing algorithms, we are able to passively identify the wireless driver employed by a device.

A number of factors affect the probing behavior of a client and make accurate fingerprinting without client cooperation a challenging task. From the perspective of an external fingerprinter, the probing behavior of a client is dependent on unobservable internal factors such as timers, and on uncontrollable external factors such as background traffic. A robust fingerprinting method cannot rely on client cooperation or assume a static environment, hence our technique uses machine learning to develop a model of a driver's behavior. This model is then used for future identification.



(a) D-Link driver for the D-Link DWL-G520 (802.11b/g) PCI wireless NIC



(b) Cisco driver for the Aironet AIR-CB21AG-A-K9 (802.11a/b/g) PCI wireless NIC

Figure 2: Plot of time delta from the previous arrival of probe request frames transmitted by two drivers.

Having explained the intuition behind our technique, we turn our attention to two examples of representative probing behavior. Figure 2(a) and Figure 2(b) are plots of the time delta between arriving probe request frames as transmitted by two different wireless drivers. Both figures clearly depict a distinctly unique cyclic pattern. We further describe the pertinent features of Figure 2(b) as a way to characterize the differences between the probing patterns. Figure 2(b) is composed of a repeating pulse with an approximate amplitude of 50 seconds. These large pulses are occasionally preceded and/or followed by much smaller pulses ranging from 1-5 seconds. These pulses indicates that probing was occurring in bursts of probe request frames sent out, on average, every 50 seconds.

Upon closer inspection, one notices that the cyclic pattern exhibited by the driver probing is characterized by small variations. Our observations reveal there are two main reasons for this. The first reason is due to loss caused by signal interference. A fingerprinter could significantly reduce this type of loss by using a higher gain antenna found on commercial grade wireless cards. The second source of variation comes from wireless drivers continuously cycling through all eleven channels in the 2.4 GHz ISM band in search of other access points. The channel cycling can be considered an additional source of loss since probe request frames transmitted on unmonitored channels cannot be observed. Multiple wireless cards could be used to monitor all eleven channels simultaneously; however, we make the more realistic assumption that a fingerprinter has a single wireless card that can only monitor a small portion (e.g. one channel at any point in time) of the eleven channels. This loss indicates that some probe requests are missed, and statistical approaches are needed to compensate for the lost frames. Given the data described above, we characterize the explicit probing behavior of a client by the sending rate of probe request frames. In the next section, we show how to leverage this characterization to accurately identify wireless drivers.

4 Device Driver Fingerprinting

The fingerprinting technique proceeds in two stages: trace capture and fingerprint generation. During trace capture, a fingerprinter within wireless transmission range of a fingerprintee captures 802.11 traffic, hereafter referred to as the trace. During fingerprint generation, the captured trace is analyzed using a supervised Bayesian approach to generate a robust device driver fingerprint.

4.1 Trace Capture

To begin the trace capture phase, we first consider how a fingerprinter might obtain a trace of probe request frames from a wireless device using widely available hardware and software. We assume a one-to-one mapping of MAC addresses to wireless devices, and believe this to be a reasonable assumption. Because each wireless NIC is assigned a unique MAC address by its manufacturer, the only cause for duplicate MACs on a network would be the result of a user reassigning his MAC address independently. However, as there are theoretically 2^{48} acceptable MAC addresses, the probability of a user choosing an existing MAC on the network is negligible[7]. In Section 7, we address the effects that violating this assumption has on our fingerprinting technique.

The fingerprinter can use any device that is capable of eavesdropping on the wireless frames transmitted by the fingerprintee. Therefore, the fingerprinter must be within receiving range of the fingerprintee's wireless transmissions. We assume the fingerprinter is using a single, high-gain, COTS (commercial off-the-shelf) wireless card. Next, the fingerprinter must configure their wireless card to operate in monitor mode; this mode allows the wireless card to capture frames promiscuously (e.g. whether they are specifically addressed to that wireless card or not). The fingerprinter must prevent their card from associating with an access point or sending its own probe request frames so collection is completely passive. This allows the fingerprinter to capture all frames sent on the current channel, including probe request frames, without interfering with the network's normal operation. We assume that the fingerprinter's machine is running an OS and driver combination that supports a wireless card in monitor mode. This can be easily done in Linux, FreeBSD, and Mac OS X. Finally, the fingerprinter can use a network protocol analyzer, such as Ethereal [6], to record the eavesdropped frames and filter out all irrelevant data. After following the above steps, the fingerprinter should have sufficient data to construct graphs similar to Figures 2(a) and 2(b).

4.2 Fingerprint Generation

After a trace has been captured, the data must be analyzed to characterize the probe request behavior. Previous work has shown that a simple supervised Bayesian approach is extremely accurate for many classification problems [8]. We chose to employ a binning approach to characterize the time deltas between probe requests because of the inherently noisy data due to frame loss.

Binning works by translating an interval of continuous data points into discrete bins. A bin is an internal value used in place of the true value of an attribute. The binning method smooths probabilities for the continuous attribute values by placing them into groups. Although binning causes some loss of information for continuous data, it allows for smooth probability estimates. Some noise is averaged out because each bin probability is an estimate for that interval, not individual con-

Bin	Percentage	Mean
0	0.676	0.16
1.2	0.228	1.72
50	0.096	49.80

Table 1: Sample signature for the Cisco Aironet 802.11 a/b/g PCI driver

tinuous values. We chose to use equal-width binning where each bin represents an interval of the same size. While more sophisticated schemes may be available, this simple approach generated distinct fingerprints of probe inter-arrival times and provided a successful means for driver identification.

After performing a number of data analysis tests, we isolated two attributes from the probing rate that were essential to fingerprinting the wireless driver. The first attribute was the bin frequency of delta arrival time values between probe request frames. The second attribute was the average, for each bin, of all actual (non-rounded) delta arrival time values of the probe request frames placed in that bin. The first attribute characterizes the size of each bin and the second attribute characterizes the actual mean of each bin. Our next step was to create a signature (Bayesian model) for each individual wireless driver that embodies these attributes. Building models from tagged data sets is a common technique used in supervised Bayesian classifiers [9].

We now describe the process used to transform raw trace data into a device signature. To calculate the bin probabilities, we rounded the actual delta arrival time value to the closest discrete bin value. For example, if the bins were of a fixed width of size 1 second, any probe request frames with a delta arrival value in (0, 0.50] seconds would be placed in the 0 second bin, any probe request frames with a delta arrival value in (0.51, 1.50] seconds would be placed in the 1 second bin, and so forth. Based on empirical optimization experiments presented in our results section, we use an optimal bin width size of 0.8 seconds. The percentage of the total probe request frames placed in each bin is recorded along with the average, for each bin, of all actual (non-rounded) delta arrival time values of the probe request frames placed in that bin. These values comprise the signature for a wireless driver which we add to a master signature database containing all the tagged signatures that are created. An example of a signature created from the probe request frames in Figure 2(b) is shown in Table 1. New signatures can be inserted, modified, or deleted from the database without affecting other signatures. This allows collaborative signature sharing, similar to how Snort [10] intrusion detection signatures are currently shared.

Once the master signature database is created, a method is required to compute how "close" an untagged signature from a probe request trace is to each of the signatures in the master signature database.

4.3 Calculating Closeness

Let us now assume that an attacker has obtained a trace and created a signature T of the probe request frames sent from the fingerprintee. Let p_n be the percentage of probe request frames in the nth bin of T and let m_n be the mean of all probe request frames in the nth bin. Let S be the set of all signatures in the master signature database and let s be a single signature within the set S. Let v_n be the percentage of probe request frames in the nth bin of s and let w_n be the mean of all probe request frames in the nth bin of s. The following equation was used to calculate the distance between the observed, untagged fingerprintee signature, T, and all known master signatures, assigning to C the distance value of the closest signature in the master database to T:

$$C = \min(\forall s \in S \sum_{0}^{n}(|p_n - v_n| + v_n|m_n - w_n|)) \quad (1)$$

Our technique iterates through all bins in T, summing the difference of the percentages and mean differences scaled by the percentage. The mean differences are scaled by the s bin percentage to prevent this value from dominating the bin percentage differences. We show in our results that the features included in a signature and our final method of calculating signature difference are effective in successfully fingerprinting wireless device drivers.

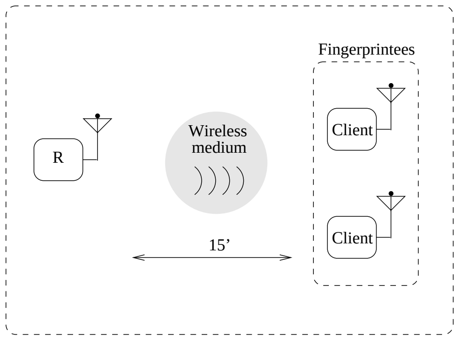
5 Evaluation

We tested our fingerprinting technique with a total of 17 different wireless interface drivers in their default configurations. We characterized wireless device drivers for the Linux 2.6 kernel, Windows XP Service Pack 1 and Service Pack 2, and Mac OS X 10.3.5. The machine we used to fingerprint other hosts' wireless drivers was a 2.4 GHz Pentium 4 desktop with a Cisco Aironet a/b/g PCI wireless card, running the Linux 2.6 kernel and the MadWifi wireless NIC driver [11]. Various Pentium III class desktop machines and one Apple PowerBook laptop were used as fingerprintee machines.

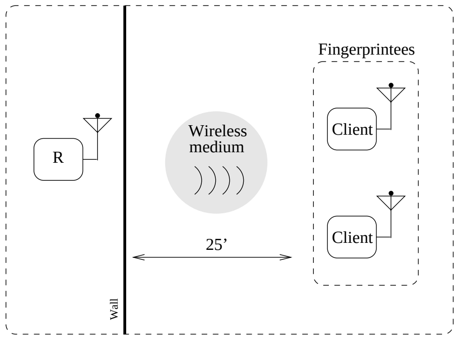
We address five primary characteristics that we expect any fingerprinting technique to be evaluated against. First, we investigate the resolution of our method. Specifically, we evaluate our identification granularity between drivers for different NICs, different drivers that support identical NICs, and different versions of the same driver. Second, we evaluate the consistency of our technique. We measure how successful our fingerprinting technique is in a variety of scenarios and over multiple network sessions, after operating system reboot, and when using the same driver to control different NICs. Third, we test the robustness of our technique. We conduct our experimentation in realistic network settings that experience loss rates similar to other wireless infrastructure networks. Fourth, we analyze the efficiency of our technique with respect to both data and time. Finally, we evaluate the resistance of our technique to varying configuration settings of a driver and evaluate the potential ways one might evade our fingerprinting technique.

To address these issues, we conducted a number of experiments using different wireless drivers and cards across a number of different operating system environments. In all cases, our technique successfully fingerprinted the wireless driver in at least one configuration. While the amount of time needed to collect the data varied across drivers and configurations, we required only a small amount of captured wireless traffic to fingerprint drivers accurately.

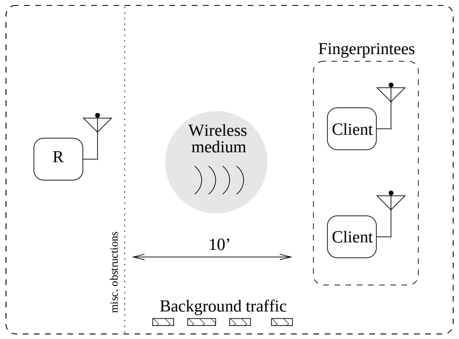
From our initial observations, we identified two properties of a device and driver that altered their signatures. The first property concerned whether the wireless device was unassociated or associated to an access point. Our initial experiments revealed that, by default, all wireless drivers transmit probe request frames when disassociated from an access point. Additionally, many continue to send probe requests even after association to an access point, though often not as frequently. The second property (only applicable to Windows drivers) concerns how the driver is managed. For many drivers, the Windows operating system can manage the configuration of the network settings for the wireless device instead of having a standalone (vendor provided) program perform those functions. The standalone program is provided by the manufacturer of the wireless device and often supports more configuration options for the specific driver, though also requires more user interaction to manage the device. We noticed slight differences in the behavior of probing depending on which option a user chose to manage their device. Due to these differences, we treated each of these property scenarios uniquely and created signatures to identify a driver under any of the appropriate cases.



(a) Test set 1 and master signature experimental setup.



(b) Test set 2 experimental setup.



(c) Test set 3 experimental setup.

Figure 3: Our test scenarios. R is the fingerprinter.

5.1 Building the Master Signatures

We collected trace data and constructed individual signatures with the same structure as the example signature in Table 1. This was repeated for all 17 wireless drivers in every configuration known to affect the signature and supported by the wireless driver. Drivers from Apple, Cisco, D-Link, Intel, Linksys, MadWifi (for Atheros chipset-based cards running under Linux), Netgear, Proxim, and SMC were included in our testing. A majority of the drivers included in our tests were for Windows; therefore most of the drivers initially had four individual signatures. We will refer to the four different configurations as follows: (1) unassociated and controlled by Windows, (2) unassociated and controlled by a standalone program, (3) associated and controlled by Windows, (4) associated and controlled by a standalone program. Three drivers did not support networking control by Windows (options 1 and 3), and four of the drivers tested did not transmit probe request frames when associated. This meant that initially, 57 signatures were compiled in the master signature database. We collected four signatures at a time and each signature trace contained a minimum of 12 hours worth of data points. A 30 minute portion of each trace was set aside and not used in signature training. This data was used as test set 1, which we further describe in the next section. As can be seen from Figure 3(a), the observing machine's antenna was placed approximately 15 feet from the fingerprintee machines, and no physical obstructions were present between the machines. Also, no 802.11 wireless traffic was detected besides the traffic generated by the fingerprintees.

After analyzing these signatures, we noted that changing configurations for some drivers had little impact on the probe request frame transmission rate and consequently, the generated signatures were indistinguishable from one another. We considered these signatures to be duplicates and removed all but one from the master signature database. This process could be automated by eliminating signatures that are insufficiently different from others with respect to some similarity threshold. There was only a single case where two of the drivers from the same manufacturer (Linksys) had indistinguishable signatures. For this case, we again left only a single signature in the master signature database. After pruning the database of all duplicate signatures, there remained 31 unique signatures. Each signature was tagged with the corresponding driver('s) name and configuration(s). The entire master signature database is included as Appendix A.

5.2 Collecting Test Data

We used the unused 30 minute trace from each of the 57 raw signature traces collected during master signature generation as test set 1. This scenario verifies that our signature generation adequately captures the probing behavior of the driver and that signatures can identify their associated drivers with a limited amount of traffic. To demonstrate that our technique is repeatable and still accurate in conditions other than where the signature data was originally collected, we repeated the 57 half hour experiments in two different physical locations. Using multiple environments helps to validate the consistency and robustness of our technique and suggests that it works well outside of lab settings. The arrangement for test set 2, as shown in Figure 3(b), was as follows: we placed the fingerprinter's antenna 25 feet from the fingerprintees with one uninsulated drywall placed in between the machines. As in Figure 3(a), no 802.11 wireless traffic was detected besides that generated by the fingerprintees. For test set 3, depicted in Figure 3(c), the observer's antenna was placed ten feet from the fingerprintees with two desks and other miscellaneous objects physically located between the machines. At this location, four to twelve other wireless devices were communicating during our data collection. Test set 2 might represent a wireless network in a semi-isolated setting, such as a hotel room with wireless access. Test set 3, on the other hand, represents a more congested wireless network, such as a network located in a coffee shop or airport.

Test Set	Successful	Total	Accuracy
1	55	57	96%
2	48	57	84%
3	44	57	77%

Table 2: Accuracy of fingerprinting technique by scenario.

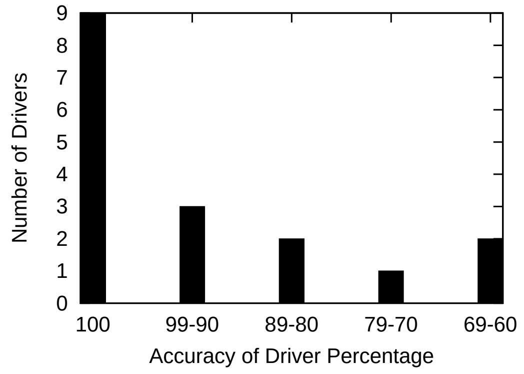


Figure 4: Number of individual drivers achieving an interval of accuracy over all test sets.

5.3 Fingerprinting Accuracy

The accuracy of our technique in correctly identifying the wireless driver operating a NIC for the three test scenarios is shown in Table 2. These results use the full half hour of data points. Later in this section, we will explore the effects of using less data points on the accuracy of our technique. The results also differed based on location. As expected, our technique is the most accurate for test set 1 (originally taken from the large signature traces) at 96%. The second most accurate test set was test set 2 (with only a single wall and no other 802.11 traffic) at 84%, and the last location had a 77% identification accuracy. These results indicate that different environments affect the accuracy of our technique. However, our technique remains reliable in all the the environments in which we tested.

Figure 4 demonstrates that our technique is perfectly accurate at fingerprinting nine of the wireless drivers and over 60% successful at identifying the other eight drivers. The accuracy of our method at identifying a particular driver is largely dependent on how dissimilar the driver's signature(s) are from other signatures in the master signature database. If the correct signature is similar to another in the database, noise such as background traffic may lead to our technique incorrectly fingerprinting a wireless driver. These results show that the majority of

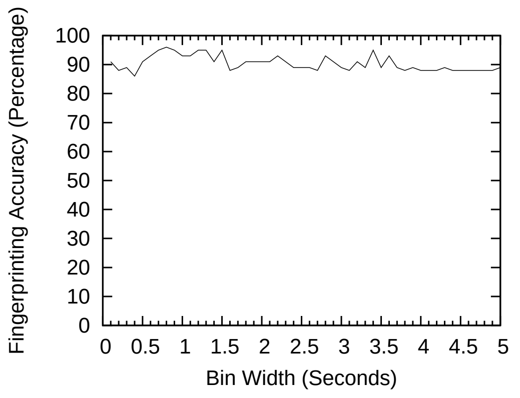


Figure 5: Empirical bin width tuning. Shows that 0.8 second wide bins generate the highest accuracy (96%) for test set 1.

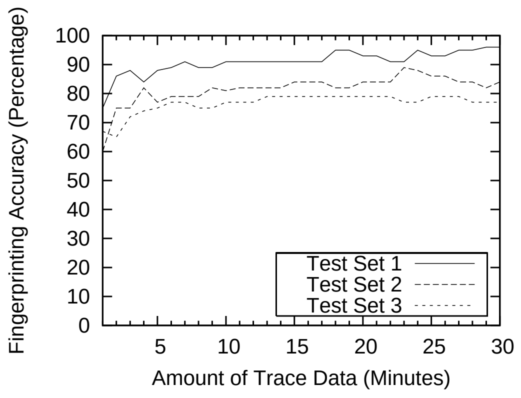


Figure 6: Effects of trace duration on fingerprinting accuracy.

wireless drivers do have a distinct signature. It is important to note that even with drivers that have less unique fingerprints, we still correctly identify the driver for a majority of the test cases.

It is also relevant to note that in cases where the technique cannot uniquely identify a driver, it was able to narrow the possibilities down to those drivers that have similar signatures. Though not supported in the current implementation of our technique, it is conceivable to list the signatures in the master signature database that are close to the unidentified observed signature.

5.4 Empirical Bin Width Tuning

The bin width for signatures was empirically optimized during our experimentation on test set 1 by varying the size in testing and selecting an optimal width based on fingerprinting accuracy. This optimization began by starting with a bin width of 0.1 seconds and incrementally increasing the bin width by 0.1 seconds up to a bin width of 5.0 seconds. Figure 5 reveals that a bin width of 0.8 seconds produced the highest accuracy (96%) in test set 1, and thus, was the bin width used for the rest of our experiments.

5.5 Time Required to Fingerprint Driver

To address our technique's efficiency, we investigated the data and time thresholds required to accurately fingerprint a driver. Ideally, a fingerprinter would be able to identify a wireless driver in real time after only a small traffic trace. We measured the fingerprinting accuracy of our method in each test scenario with one minute of collected data and increased the amount of data in one minute increments until the full thirty minute trace from each setting was used. Figure 6 illustrates the accuracy of our technique in each of the three test cases corresponding to the amount of trace data used for fingerprinting.

Since the rate of probe request frames is different for most wireless drivers, it is difficult to estimate how many probe request frames will be recorded during one minute of observation, though for statistical interest, the average number of probes detected during one minute of observation was 10.79 across all of our testing scenarios. The accuracy of our technique is at least 60% in each of the three test cases after only one minute of traffic. These results show that our method successfully converges relatively fast on the correct wireless driver and needs only a small amount of communication traffic to do so.

6 Limitations

In the course of our evaluation, we discovered a few limitations of our fingerprinting technique. We discuss these in detail below.

6.1 Driver Versions

One of the original questions we posed concerned the resolution of our technique. We have shown that our technique is capable of distinguishing between different drivers the vast majority of the time. We are also interested in whether our method can distinguish between two different versions of the same wireless driver. A number of wireless card manufactures have released new versions of their wireless drivers to support new features. We tested our fingerprinting technique on six wireless drivers, with multiple driver versions available to determine if it was possible to distinguish between different versions of the same wireless driver. Our technique was unsuccessful in distinguishing between different versions of the same driver. This is a limitation of our fingerprinting technique since a new version of a driver might patch previous security vulnerabilities in the driver. However, even without the ability to distinguish between versions, our fingerprints greatly reduce the number of potential wireless drivers that a target system is running.

6.2 Hardware Abstraction Layer

Another unexpected limitation was found when testing the MadWifi driver for Linux. This driver works with most wireless cards containing the Atheros chipset because of the inclusion of a Hardware Abstraction Layer (HAL). This creates a more homogeneous driver environment since a majority of wireless cards currently available use the Atheros chipset. The side effect is that the lack of driver diversity reduces the appeal of fingerprinting wireless drivers. However, one drawback of a single (or relatively small number of) hardware abstraction layer(s) is that it magnifies any security vulnerability identified.

7 Preventing Fingerprinting

Several methods can be used to prevent our technique from successfully fingerprinting drivers. These methods include configurable probing, standardization, automatic generation of noise, driver code modification, MAC address masquerading, and driver vulnerability patching.

7.1 Configurable Probing

One solution to prevent our fingerprinting technique is for device drivers to provide the option to explicitly disable or enable probe request frames. It makes sense for this to be a configurable option not only to prevent fingerprinting but also to conserve power and bandwidth. Probe request frames are used to find networks matching the available data rates on the client device [7]. The SSID of the desired network can be specified or can be set to the broadcast SSID when probing for any available networks. By default, access points transmit beacon frames, which announce the access point's presence and some configuration information[8]. Thus, passively listening for beacons (i.e., turning off probe request frames) could be an effective method of discovering access points. Another solution would be to configure wireless device drivers, by default, to passively listen for beacons and only send probe requests for available networks when manually triggered by the user.

7.2 Standardization

An effective, but potentially difficult to implement solution for preventing driver fingerprinting is to specify the rate at which probe request frames are transmitted in a future IEEE standard for the 802.11 MAC. Another step towards standardization could result if a corporate body or open source consortium was formed to develop a standard agreed upon by all driver manufactures. If all driver manufactures adhered to such a standard, the described fingerprinting method would be rendered useless. Unfortunately, there are many obstacles preventing such a standard, the major factor being that some device manufacturers will not want to design devices that expend the power or bandwidth necessary to transmit probe requests at a standard rate. Due to this reason alone, it is doubtful that there will be any standardization agreed upon and followed by every driver manufacture concerning the rate of probe request frame transmission.

7.3 Automated Noise

Another strategy to prevent wireless driver fingerprinting is to generate noise in the form of cover probe request frames. Cover traffic disguises a driver by masking the driver's true rate of probe request transmission. Due to the fact that our technique uses statistical methods to filter out noise, the cover traffic would need to be sufficiently random and transmit enough cover to confuse our technique. A limitation of this approach is that the cover probe request frames waste bandwidth the device would otherwise use for wireless traffic, and for devices with limited power supplies, transmitting cover traffic would reduce battery life significantly. Also, given enough observation data, the fingerprinter might be able to filter away the noise and successfully fingerprint the driver. Generating noise is a difficult problem as many data mining algorithms have been shown to be effective in filtering out such noise and recovering the original data [12, 13, 14].

7.4 Driver Code Modification

For open source drivers such as the Madwifi drivers, the driver code could be modified to change the transmission rate of probe request frames. This alteration would fool our fingerprinting technique. However, this is only possible for open source drivers and would require a skilled programmer to alter the driver code. This would not be possible for many windows drivers, since most do not provide source code.

7.5 MAC Address Masquerading

Earlier, we made the assumption of a one-to-one mapping of MAC addresses to wireless devices. One method to prevent driver fingerprinting is to change the device's MAC address to match the MAC address of another device within transmission range. This would fool our fingerprinting technique into believing probe requests from two different wireless drivers are originating from the same wireless driver. There are a number of problems with this solution. First, the wireless device must make certain that the fingerprinter is within transmission range of both wireless devices. If the fingerprinter only observes probe request frames from one of the two devices, it will not be deceived. Also, since our method uses statistical methods to filter noise, the wireless device needs to make certain that the other device is transmitting enough probe request frames to mask its signature.

7.6 Driver Patching

While driver patching is not a full solution, we feel the creation of well thought out driver patching schemes would improve the overall security of device drivers as new driver exploits are found. Current research is being conducted to improve the process of patching security vulnerabilities [15, 16]. The device driver community should leverage this research to create more robust patching methods, and improve the overall level of driver security.

8 Related Work

Various techniques for system and device level fingerprinting have been used for both legitimate uses, such as forensics and intrusion detection, as well as malicious uses, such as attack reconnaissance and user profiling. The most common techniques take advantage of explicit content differences between system and application responses. Nmap [17], p0f [18], and Xprobe [19] are all open source, widely distributed tools that can remotely fingerprint an operating system by identifying unique responses from the TCP/IP networking stack. As the TCP/IP stack is tightly coupled to the operating system kernel, these tools match the content of machine responses to a database of OS specific response signatures. Nmap and Xprobe actively query the target system to invoke these potentially identifying responses. In addition to this active probing, p0f can passively fingerprint an operating system by monitoring network traffic from a target machine to some third party and matching characteristics of that traffic to a signature database. Data link layer content matching can also be used to identify wireless LAN discovery applications [20], which can be useful for wireless intrusion detection.

While datagram content identification methods are arguably the most simple, they are also limited to situations where datagram characteristics are uniquely identifiable across systems, as well as accessible to an outside party. Except for a few unique instances, 802.11 MAC-layer frame formatting and content is generally indistinguishable across wireless devices; because of this, more sophisticated methods are often required. In [21], the authors present a technique to identify network devices based on their unique analog signal characteristics. This fingerprinting technique is based on the premise that subtle differences in manufacturing and hardware components create unique signaling characteristics in digital devices. While the results of analog signal fingerprinting are significant, this method requires expensive hardware such as an analog to digital converter, IEEE 488 interface card, and digital sampling oscilloscope. Also, it is not clear from their analysis of wired Ethernet devices whether this method would be feasible in a typical wireless network setting where noise from both the environment and other devices is a more pressing consideration.

A device's clock skew is also a target for fingerprinting. A technique presented in [22] uses slight drifts in a device's TCP option clock to identify a network device over the Internet via its unique clock skew. Whereas our technique fingerprints which driver a wireless device is running, time skew fingerprinting is used to identify distinct devices on the Internet. Concerning security, unique device fingerprinting is often not as useful as driver and other types of software fingerprinting. As opposed to content based fingerprinting, both analog signal and time skew fingerprinting exploit characteristics of the underlying system hardware, making these techniques much more difficult to spoof.

Identification via statistical timing analysis in the context of communication patterns and data content has been especially studied in the area of privacy enhancing technologies. While network security mechanisms such as encryption are often utilized to protect user privacy, traffic analysis of encrypted traffic has proven successful in linking communication initiators and recipients participating in anonymous networking systems [23, 24]. Traffic analysis has also been applied to Web page fingerprinting. In [25], the authors demonstrate a technique that characterizes the inter-arrival time and datagram sizes of web requests for certain popular web sites. Using these web page characterizations, one can identify which sites users on wireless LANs are visiting despite these users browsing the Internet via encrypted HTTP traffic streams.

The techniques described above serve as only a survey of existing fingerprinting techniques for systems, devices, and even static content. The approaches vary from exploiting content anomalies in the TCP/IP stack to characterizing time-based system behavior at both the physical and software layers of a system. While the approaches vary, these contributions bring to light the true feasibility of fingerprinting via avenues otherwise assumed to be uniformly implemented across systems.

9 Conclusion

We designed, implemented, and evaluated a technique for passive wireless device driver fingerprinting that exploits the fact that most IEEE 802.11a/b/g wireless drivers have implemented different active scanning algorithms. We evaluated our technique and demonstrated that it is capable of accurately identifying the wireless driver used by 802.11 wireless devices without specialized equipment and in realistic network conditions. Through an extensive evaluation including 17 wireless drivers, we demonstrated that our method is effective in fingerprinting a wide variety of wireless drivers currently on the market. Finally, we discussed ways to prevent fingerprinting that we hope will aid in improving the security of wireless communication for devices that employ 802.11 networking.

10 Acknowledgments

Some of this work was performed while the authors were at Sandia National Laboratories - California. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000. The authors greatly appreciate the staff of Sandia National Laboratories for their assistance. The authors would like to thank John Bethencourt, Nikita Borisov, Frank Hemingway, Adam Lee, Kristen Pelon, Amanda Stephano, and the anonymous reviewers for their useful suggestions. This work was partially supported by NSF Grant ITR-0428887 (Spectrum Management Toward Spectrum Plenty) and the University of Colorado. Jason Franklin performed this research while on appointment as a U.S. Department of Homeland Security (DHS) Fellow. The views expressed in this paper do not necessarily reflect the policies and views of DHS, DOE, or affiliated organizations.

References

[1] Ken Ashcraft and Dawson R. Engler. Using Programmer-Written Compiler Extensions to Catch Security Holes. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2002.

[2] Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson R. Engler. An Empirical Study of Operating System Errors. In *Proceedings of Symposium on Operating Systems Principles (SOSP 2001)*, October 2001.

[3] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proceedings of Symposium on Operating Systems Principles (SOSP 2003)*, October 2003.

[4] IEEE-SA Standards Board. IEEE Std IEEE 802.11-1999 Information Technology - Wireless LAN Medium Access Control (MAC) And Physical Layer (PHY) Specifications. IEEE Computer Society, 1999.

[5] IEEE-SA Standards Board. Amendment 6: Medium Access Control (MAC) Security Enhancements. IEEE Computer Society, April 2004.

[6] Ethereal: A network protocol analyzer. Web site, 2006. ⟨`http://www.ethereal.com`⟩.

[7] Matthew S. Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2nd edition, 2005.

[8] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[9] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[10] Snort Intrusion Detection and Prevention system. Web site, 2006. ⟨`http://www.snort.org/`⟩.

[11] Madwifi: Atheros chip set drivers. Web site, 2006. ⟨`http://sourceforge.net/projects/madwifi/`⟩.

[12] D. Agrawal and C. C. Aggarwal. On the Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proceedings of Symposium on Principles of Database Systems*, 2001.

[13] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of ACM SIGMOD*, May 2000.

[14] B. Hoh and M. Gruteser. Location Privacy Through Path Confusion. In *Proceedings of IEEE/CreateNet International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, 2005.

[15] Gautam Altekar, Ilya Bagrak, Paul Burstein, and Andrew Schultz. OPUS: Online Patches and Updates for Security. In *Proceedings of 14th USENIX Security Symposium*, Aug 2005.

[16] John Dunagan, Roussi Roussev, Brad Daniels, Aaron Johnson, Chad Verbowski, and Yi-Min Wang. Towards a Self-Managing Software Patching Process Using Black-Box Persistent-State Manifests. In *First International Conference on Autonomic Computing (ICAC'04)*, 2004.

[17] Nmap: a free network mapping and security scanning tool. Web site, 2006. ⟨`http://www.insecure.org/nmap/`⟩.

[18] Project details for p0f. Web site, 2004. ⟨`http://freshmeat.net/projects/p0f/`⟩.

[19] Arkin and Yarochkin. Xprobe project page. Web site, August 2002. ⟨`http://sourceforge.net/projects/xprobe`⟩.

[20] Joshua Wright. Layer 2 Analysis of WLAN Discovery Applications for Intrusion Detection. Web site, 2002. ⟨`http://www.polarcove.com/whitepapers/layer2.pdf`⟩.

[21] Ryan Gerdes, Thomas Daniels, Mani Mina, and Steve Russell. Device Identification via Analog Signal Fingerprinting: A Matched Filter Approach. In *Proceedings of the Network and Distributed System Security Symposium Conference (NDSS 2006)*, 2006.

[22] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (SP 2005)*, Washington, DC, USA, 2005.

[23] Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Proceedings of Privacy Enhancing Technologies Workshop (PET 2000)*, May 2000.

[24] Mathewson and Dingledine. Practical Traffic Analysis: Extending and Resisting Statistical Disclosure. In *Proceedings of Privacy Enhancing Technologies Workshop (PET 2004)*, May 2004.

[25] George Dean Bissias, Marc Liberatore, and Brian Neil Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Proceedings of Privacy Enhancing Technologies Workshop (PET 2005)*, May 2005.

[26] Mike Kershaw. Kismet. Web site, 2006. ⟨`http://www.kismetwireless.net/`⟩.

Appendix A

This appendix includes the entire master signature database from our evaluation section. It is organized with the name of the wireless driver, if the driver was associated (assoc) or unassociated (unassoc), and if Windows (win) was configuring the wireless device, or a standalone program (native). The values after the driver name and configuration are a set of tuples ordered as follows: (Bin Value, Percentage, Bin Mean Value).

cisco-abg-assoc-native (0.8,0.101,0.677)(1.6,0.108,1.450) (2.4,0.168,2.377)(3.2,0.021,2.928)(4,0.024,3.798) (4.8,0.028,4.691)(5.6,0.048,5.536)(6.4,0.034,6.303) (7.2,0.080,7.132)(8,0.032,7.830)(8.8,0.017,8.473) (9.6,0.044,9.607)(53.6,0.288,53.399)

cisco-abg-unassoc-native (0,0.514,0.048)(0.8,0.285,0.749) (1.6,0.037,1.656)(2.4,0.028,2.373)(3.2,0.067,3.264) (4.8,0.041,4.981)(5.6,0.025,5.521)

cisco-abg-unassoc-win (0,0.466,0.072)(0.8,0.126,0.720)(1.6,0.115,1.479)(2.4,0.056,2.345)(3.2,0.040,3.089)(4,0.025,3.843)(4.8,0.020,4.592)(5.6,0.019,5.415)(6.4,0.012,6.129)(8.8,0.013,8.554)(49.6,0.063,49.639)(50.4,0.026,50.146)

dwl-ag530-assoc-native (0,0.420,0.032)(0.8,0.108,0.590)(1.6,0.043,1.358)(2.4,0.011,2.067)(4.8,0.113,4.470)(5.6,0.060,5.477)(6.4,0.039,6.192)(7.2,0.030,7.206)(8,0.011,7.630)(12,0.010,11.829)(51.2,0.144,50.995)

dwl-ag530-unassoc-native (0,0.544,0.034)(0.8,0.052,0.597)(1.6,0.198,1.670)(6.4,0.053,6.659)(7.2,0.129,7.248)(8,0.012,7.806)

dwl-ag650-assoc-win (0,0.392,0.008)(0.8,0.231,0.549)(1.6,0.049,1.481)(2.4,0.030,2.416)(3.2,0.045,3.250)(4,0.067,4.092)(4.8,0.021,4.687)(58.4,0.164,58.198)

dwl-ag650-unassoc-win (0,0.606,0.084)(0.8,0.233,0.621)(1.6,0.090,1.689)(2.4,0.068,2.322)

dwl-g520-unassoc-native (0,0.533,0.054)(0.8,0.246,0.674)(1.6,0.072,1.541)(2.4,0.035,2.539)(3.2,0.079,2.989)(4,0.026,3.706)

dwl-g520-unassoc-win (0,0.527,0.055)(0.8,0.236,0.666)(1.6,0.134,1.523)(2.4,0.039,2.401)(3.2,0.044,3.109)(4,0.015,3.791)

engenuis-unassoc-win (0,0.193,0.059)(0.8,0.104,1.188)(1.6,0.609,1.271)(2.4,0.082,2.529)(4,0.011,3.814)

intel2100-assoc-win (0,0.766,0.019)(63.2,0.234,62.949)

intel2100-unassoc-win (0,0.927,0.055)(30.4,0.073,30.132)

intel-2200-assoc-native (0,0.591,0.107)(0.8,0.071,0.955)(1.6,0.079,1.495)(2.4,0.107,2.182)(120,0.050,120.254)(120.8,0.091,120.698)

intel-2200-unassoc-native (0,0.659,0.078)(0.8,0.015,0.882)(32.8,0.031,33.063)(34.4,0.139,34.765) (35.2,0.142,34.853)

intel-2915-assoc-native (0,0.659,0.080)(0.8,0.032,0.938)(1.6,0.037,1.426)(118.4,0.171,118.155) (119.2,0.076,119.193)

intel-2915-unassoc-native (0,0.668,0.083)(32.8,0.331,32.868)

linksys-pci-unassoc-win (0,0.348,0.165)(0.8,0.273,0.923)(1.6,0.032,1.262)(61.6,0.262,61.787)(62.4,0.027,62.270)(63.2,0.054,62.953)

madwifi-unassoc (72.8,0.881,72.988)(133.6,0.119,133.978)

netgear-assoc-win (0,0.423,0.001)(0.8,0.203,0.611)(1.6,0.038,1.552)(2.4,0.058,2.240)(3.2,0.037,3.206)(4,0.016,4.006)(4.8,0.060,4.731)(5.6,0.010,5.505)(57.6,0.149,57.498)

netgear-unassoc-win (0,0.560,0.061)(0.8,0.135,0.652)(1.6,0.077,1.532)(2.4,0.018,2.340)(3.2,0.023,3.125)(4,0.106,4.035)(4.8,0.071,4.566)

osx-airportb-unassoc (0,0.639,0.022)(10.4,0.361,10.295)

proxim-assoc-native (0,0.035,0.396)(0.8,0.377,0.585)(1.6,0.133,1.376)(2.4,0.016,2.078)(4.8,0.168,4.523)(5.6,0.035,5.535)(55.2,0.087,55.400)(56,0.024,56.017)(56.8,0.084,56.836)(57.6,0.022,57.435)

proxim-assoc-win (0,0.039,0.385)(0.8,0.329,0.585)(1.6,0.118,1.385)(2.4,0.020,2.055)(4.8,0.089,4.681)(5.6,0.089,5.500)(6.4,0.032,6.242)(7.2,0.013,7.167)(55.2,0.122,55.402)(56,0.036,56.037)(56.8,0.068,56.773)(57.6,0.012,57.466)

proxim-unassoc-win (0,0.540,0.052)(0.8,0.229,0.660)(1.6,0.090,1.555)(2.4,0.012,2.328)(4.8,0.055,5.011)(6.4,0.040,6.479)

smc-2532w-assoc-native (0,0.619,0.140)(0.8,0.028,0.477)(1.6,0.013,1.812)(60.8,0.013,60.907)(62.4,0.183,62.595)(63.2,0.118,62.899)

smc-2532w-unassoc-win (0,0.065,0.140)(0.8,0.047,0.727)(1.6,0.880,1.681)

smc-2632w-unassoc-native (0,0.511,0.083)(10.4,0.470,10.555)

smc-wpci-assoc-native (0,0.461,0.001)(0.8,0.117,0.588)(1.6,0.139,1.678)(2.4,0.020,2.185)(4,0.021,3.915)(4.8,0.093,5.028)(56,0.127,55.708)

smc-wpci-unassoc-native (0,0.563,0.038)(0.8,0.144,0.689)(1.6,0.014,1.790)(4,0.093,3.857)(4.8,0.079,4.952)(5.6,0.057,5.935)(6.4,0.026,6.178)

wpc54g-assoc-win (0,0.633,0.038)(0.8,0.114,0.437)(62.4,0.148,62.550)(63.2,0.105,62.981)

wpc54g-unassoc-native (0,0.623,0.054)(0.8,0.151,0.633)(62.4,0.172,62.299)(63.2,0.055,62.960)

Notes

[1]Carnegie Mellon University,`jfrankli@cs.cmu.edu`

[2]University of Colorado, Boulder, `damon.mccoy@colorado.edu`

[3]University of Illinois, Urbana-Champaign, `tabriz@uiuc.edu`

[4]University of California, Davis,`vneagoe@ucdavis.edu`

[5]Sandia National Laboratories,`jvanran@sandia.gov`

[6]University of Colorado, Boulder,`douglas.sicker@colorado.edu`

[7]It is important to note that some attackers will sniff the MAC addresses of other users on a wireless network to use as their own, giving them the ability to steal a connection or hide their malicious actions. Although we acknowledge that this scenario would bring about duplicate MAC addresses on a network, we believe it is far from the common case in most network settings.

[8]This is in contrast to disabling the SSID broadcast function. Disabling SSID broadcast simply forces an AP to send a string of spaces or a null string in the SSID field of the beacon frame. Kismet [26] reports this SSID as `<no ssid>`.