3-25-2008

Active Behavioral Fingerprinting of Wireless Devices

Sergey Bratus
Dartmouth College

Cory Cornelius
Dartmouth College

David Kotz
Dartmouth College

Daniel Peebles
Dartmouth College

Active Behavioral Fingerprinting of Wireless Devices

Dartmouth Computer Science Technical Report TR2008-610

Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles
Institute for Security Technology Studies
Dartmouth College
Hanover, New Hampshire, USA

March 25, 2008

Abstract

We propose a simple active method for discovering facts about the chipset, the firmware or the driver of an 802.11 wireless device by observing its responses (or lack thereof) to a series of crafted non-standard or malformed 802.11 frames. We demonstrate that such responses can differ significantly enough to distinguish between a number of popular chipsets and drivers. We expect to significantly expand the number of recognized device types through community contributions of signature data for the proposed open fingerprinting framework. Our method complements known fingerprinting approaches, and can be used to interrogate and spot devices that may be spoofing their MAC addresses in order to conceal their true architecture from other stations, such as a fake AP seeking to engage clients in complex protocol frame exchange (e.g., in order to exploit a driver vulnerability). In particular, it can be used to distinguish rogue APs from legitimate APs before association.

Chapter 1

Introduction

In this paper we consider *fingerprinting* of different 802.11 MAC layer implementations in wireless devices, that is, methods for discovering facts about the chipset, the firmware or the driver of the device without relying on the information contained in the manufacturer code part of the device's MAC address or other information that the device may provide about itself. In particular, we focus on *active* fingerprinting that relies on exchanging crafted 802.11 frames with a device and observing its responses. For this purpose, an active fingerprinting platform emulates (most often with modifications) parts of standard 802.11 protocol exchanges in order to elicit characteristic responses that are known to differ between different implementations.

Tools for *fingerprinting* networked devices, i.e., discovering facts about their software, OS or hardware that their operators may want to conceal or misrepresent, are nowadays an indispensable part of security assessment and penetration testing toolkits, and are routinely included in popular GNU/Linux and other security-conscious OS distributions. They help administrators to inventory and penetration testers to reconnoiter networks; they are used to locate vulnerable systems and unauthorized devices; they are useful for interrogating and outing suspected masquaraders. These tools, however, operate at the network layer (L3) and transport layer (L4) exclusively.

In the past few years, fingerprinting of wireless stations has been recognized as an important application and attracted attention from different angles, for reasons we discuss later.

1.0.1 802.11 L2 attacks and offensive fingerprinting

Although, in theory, all complicated computer systems are likely to be "equal" in the sense of being vulnerable to attacks, from an attacker's point of view some of them are definitely "more equal than others". This has ensured the interest in various target fingerprinting techniques, both active and passive, and their implementation in a variety of widely popular tools.

Several vulnerabilities recently discovered in different implementations of the 802.11 MAC layer resulted in increased interest in fingerprinting 802.11 platforms for both offensive and defensive uses. The Blackhat 2006 public demonstration of a kernel-level exploit for an 802.11 card driver has validated predictions that the complexity of 802.11 protocols meant exploitable vulnerabilities and generated significant amounts of publicity. This release was accompanied by reports of successful fingerprinting of hardware-software combinations, based on their link-layer behavior, as a means of conducting precisely targeted attacks [4]. Shortly after, the so-called "Month of kernel bugs" (see, e.g., [2]) further proved the point. Suddenly, targeted attacks on 802.11 platforms became a frightening reality.

The most obvious reason for this is that the majority of existing exploits are still developed to target a particular platform.[1] Other circumstances driving this development are the relative complexity of standard

[1] The announcement of exploit code techniques that produce code binary compatible with two or more different architectures,

protocols (802.11 stands out in this respect, as one of the most complex lower level protocols) and the fact that the vendor differences in their implementation produced behaviors liable to be detected or exploited by attackers.

1.0.2 An inspiration: TCP/IP fingerprinting

The current situation somewhat resembles that of TCP/IP stacks of the 90s that used to fall to attacks as trivial as the infamous "land", "bonk" and "teardrop".[2] It is further exacerbated by the fact that, in the case of 802.11, products were widely released before many fine points of the protocol were specified, as vendors were driven to "differentiate" their products rather than achieve interoperability.[3]

The need for testing TCP/IP stack implementations (in particular, for hard to mitigate kernel-level vulnerabilities) led to development of test suites such as PROTOS [1]. Differences in the implementation of TCP/IP stacks also presented ample opportunities for fingerprinting. In particular, IP fragmentation and features of UDP and TCP have been observed to allow fingerprinting IP stacks by vendor and by OS version and become the basis of classical tools, e.g., *nmap*, *xprobe2*, and the passive *p0f*. They were also shown to have major implications for network intrusion detection (e.g., Ptacek and Newsham [11]). Widespread offensive use of target stack fingerprinting techniques created interest in thwarting such reconnaissance as a defensive measure for vulnerable TCP/IP stacks (e.g., Provos [10] and Wang [13]).

These examples and tools serve as an inspiration for our fingerprinting approach.

With the emergence of 802.11 driver attacks and platform-specific attacks (not necessarily on the link layer itself, but targeting other platform-specific software) offensive use of fingerprinting will, no doubt, increase. We expect mobile systems, including Wi-Fi appliances and wireless phones, to be particularly vulnerable, in part, due to market pressures affecting the mobile software-development process and the added complexities of patching mobile systems (see, e.g., the analysis of mobile devices' vulnerabilities in [9]). We note that even simple denial-of-service attacks on mobile devices can be more useful to the attacker, because they may cause the user to reset the device and thus repeat various sensitive negotiation exchanges, which are usually assumed by the designers to occur in non-hostile environments (e.g., Bluetooth pairing).

1.0.3 Fingerprinting for defense

Fingerprinting tools have a long tradition of defensive uses by network defenders and system administrators, such as inventorying the network and locating unauthorised devices. Far from being exclusively attacker's tools, they are routinely included as administrators' tools in popular GNU/Linux and other OS distributions distributions.

We consider two basic use scenarios for defensive wireless fingerprinting. In the first scenario, suppose that the owner of a client station either lacks the cryptographic capabilities necessary to authenticate the access point and is concerned that the latter may be an "evil twin" set up to engage the user's station in a man-in-the-middle attack, or, even though strong authentication of the AP is possible in later stages, is concerned that an "evil twin" may be seeking to compromise it via a driver or supplicant vulnerability even before the cryptographic verification is complete.

Thus the user may want to gain additional information by fingerprinting the AP before further communicating with it. Since APs by design respond to a wider range of stimuli than stations, this scenario naturally fits active fingerprinting. Moreover, if the feared attack is a targeted one, the user may entirely lack the opportunity to perform passive fingerprinting, since the "evil twin" may not provide such an opportunity to profile itself.

The second scenario is that of a network administrator concerned with keeping unauthorized equipment from his network, such as cheap commodity access points, which employees bring in without permission, testing or

although exciting, is not likely to change this general tendency.

[2] Land: `http://insecure.org/sploits/land.ip.DOS.html`,
Bonk: `http://insecure.org/sploits/95.NT.fragmentation.bonk.html`,
Teardrop: `http://insecure.org/sploits/linux.fragmentation.teardrop.html`

[3] We are indebted to an anonymous reviewer for pointing out this important circumstance.

configuration assessment. Even if an employee introducing such a device changes its wireless interface's MAC address and other settings, active fingerprinting is likely to expose it as being of different architecture than the approved devices.

A variant of this scenario could be that of an administrator desiring to ensure that only certain configurations of mobile stations are allowed to interact with the network, and is concerned that users might replicate the credentials and the MAC address of an allowed device on a device of different architecture. Engaging the device in an active fingerprinting conversation to provide an extra degree of assurance that it is what it claims to be may then be useful. In this case the administrator can even configure his APs to ignore the device (and thus limit their potential exposure to hostile behavior) until it passes some fingerprinting tests.

Our active fingerprinting method can be used in the above scenarios to help protect a vulnerable client from a rogue access point attempting to masquerade as a part of the approved infrastructure long enough to exploit the client before it fails a credentials check on its way to association, or vice versa.

Chapter 2

Related work and our approach

There exists a wide variety of physical layer fingerprinting methods, including signal, timing, and frequency analysis, capable of fingerprinting individual radio devices well beyond their general architecture.

We take a much more limited approach by concentrating on 802.11 L2 implementation differences and behaviors that

(a) can be observed with a commodity card in RF monitor mode, and

(b) do not require passive observation of existing connections, but can instead be

(c) reliably elicited by sending (preferably few) stimulus frames that commodity cards can inject (possibly under patched drivers).

In short, we are limiting ourselves to a kit accessible to a dedicated network administrator.

Works meeting the first requirement (a) include passive client station fingerprinting techniques [6], and an empirical analysis of heterogeneity 802.11 MAC implementations [7], which mentions the possibility of fingerprinting by observing characteristics such as random backoff interval calculation, and handling of power management.[1] In our opinion, the Cache [3] method for driver and chipset fingerprinting based on observed values of the Duration field, deserves a special mention for demonstrating that in a MAC layer as complex as 802.11 even a single 16 bit integer field can provide enough information about a particular implementation.

In this paper we explore a complementary *active* approach to fingerprinting, based on responses of 802.11 MAC implementations to malformed and non-standard frames. Our goal was to identify the simplest fingerprintable behaviors that could be elicited by sending such frames to APs and stations and used to distinguish between chipset and the software or firmware involved.

Although we plan to use statistical fingerprints in our future work, in this paper we further narrowed the scope to non-statistical behaviors such as differing responses (or the lack thereof) to certain flag bits set or cleared, and to other fields having unusual values. We show that such reactions may be enough to recognize devices providing wrong information about their manufacturer, chipset, etc.

This *active* approach to fingerprinting has both inherent strengths and weaknesses. Unlike passive approaches, it may provide a negative answer, i.e., give reasonable assurances that a given implementation is *not* what it claims to be, much faster than the passive ones, in the best cases after exchanging only a few stimulus frames with the target. On the other hand, it is “noisy” by definition and thus more likely to draw the attention of a wireless IDS with its stimulus frames and might even crash some vulnerable devices.

Offensive uses of active fingerprinting are thus limited to scenarios where the attacker does not care about immediate detection of his transmissions by either a rule-based or an anomaly-based link-layer aware WIDS.

[1] The authors also note the use of vendor-specific information elements in beacons and point out that these can be used to identify the manufacturer of the AP. However, faking them is not hard either.

Chapter 3

Method

In this chapter describe our active fingerprinting approach. We started with the hypothesis that different implementations of 802.11b/g would react differently to non-standard events, and, perhaps, in the interest of performance, even cut corners when sanity-checking frames that require a fast response.

To structure our tests, we charted all possible combinations of features derived from the fields of the 802.11 MAC header, and marked non-standard or unusual combinations. We collected and classified these combinations in Table 7.1.

In particular, we singled out combinations of frame types, subtypes and Frame Control flags either prohibited by the standard, not explicitly prohibited but of unclear utility for the standard protocols, or simply unlikely to occur in practice. We then drafted a number of scenarios unlikely to occur in a normal network and built *BAFFLE*,[1] a generator and injector of non-standard and malformed frames. Our injection framework uses the *LORCON* library.[2] We use a custom tool written in the Ruby programming language for shaping the frames. Its design was informed by the *Scapy* tool.[3]

3.1 Stimulus-response approach

To fingerprint a wireless station we use a series of tests, the outcomes of which are combined in a decision tree structure. We describe an automated method for constructing decision trees in Section 5. Such a method will become necessary when the database of tests reaches significant size, as we plan to publish our testing tools and incorporating contributed test results. Currently the decision tree can easily be managed manually.

Each test involves sending a "stimulus" frame to a station and observing the response, which is then tested, and the outcome of the test determines transition to the next decision tree node. The responses considered are of the following types:

(a) any frame transmitted by the station within timeout of the stimulus,

(b) a frame of specific type and subtype transmitted by the station within a given time-out period, and

(c) lack of any transmitted frames for the given time-out period.

Due to the asymmetric nature of the interactions between a station and the access point, the chosen scenarios fall into two separate classes for APs and client stations.

[1] BAFL stands for Behavioral Active Fingerprinting of the Link (layer); the rest of the name suggested itself after several devices we tested became baffled by our stimuli and required hard reboots.

[2] http://www.802.11mercenary.net/lorcon/

[3] http://www.secdev.org/projects/scapy/

3.2 Fingerprinting an access point

On an open 802.11 network without additional cryptographic authentication, client stations lack the capability to authenticate an access point. Therefore fingerprinting an access point can serve as additional assurance for the mobile user that she is not dealing with an "evil twin".

We identified the following active fingerprinting scenarios:

1. The FromDS and ToDS bits on a Probe Request are expected to be cleared. We send Probe Request frames with one or both of these bits set. Some APs (most notably Linksys WRT54g[4] responded to such Probe Requests with their usual Probe Response, whereas others did not.

2. Responses to Probe Requests with other Frame Control bits set (in particular, More Fragments, Power Management, More Data, and Order bits) differed between APs.

3. The FromDS and ToDS bits are expected to be cleared on Authentication Requests. Other Frame Control bits are also expected to be cleared on Authentication Requests.

4. As a variant of the previous two tests, the same is applied to the Association Requests after a successful Authentication Request – Authentication Response exchange.

5. Probe Requests and Authentication Requests are not expected to be fragmented. The AP's reaction to a fragmented request may vary and thus be fingerprintable.

6. A Probe Request is expected to contain certain information elements such as those with the ESSID and required rates. The reaction to Probe Requests without such elements may vary.

7. Once a station is associated with an AP that supports power saving, it can notify the AP with a Null Data frame with the PS bit set. Other bits in the Frame Control field are not expected to be set for such requests; the AP's response to a Null Data frame with these bits set may vary.

3.3 Fingerprinting a client station

Fingerprinting a station based on its responses may be desirable when an open WLAN is expected to have only a limited number of connecting stations using approved hardware. It can also be used to discover unexpected "substitutions" of one physical station for another (e.g., as a result of interception or sharing of credentials). Many sites still use MAC registration of Wi-Fi interfaces for keeping track of users if not for actual access control; they might want to occasionally check if any station substitution has occurred.

Due to the above mentioned asymmetry between the station and the AP, stations' reactions are easiest to observe when a station is either probing or is in the authenticated and associated state. The stimuli for such responses are likely to be more disruptive than those for the AP outlined above, since they induce (or fail to induce) association or authentication state changes. A well-written driver and network stack should, however, be able to restore the association without interrupting connections at the higher layers.

As a baseline, we must first measure the reaction of the station to well-formed deauthentication and deassociation frames that are meant to induce state transitions. A typical client station attempts re-association and re-authentication as per the standard 802.11 state machine. Note that:

1. Deauthentication and Deassociation frames are not expected to have certain Frame Control bits set (e.g., FromDS and ToDS). Stations may not react to malformed injected frames and continue in associated state.

[4] We tested versions 2, 6, and 8, which yielded very similar results.

2. The station's response to undefined reason codes in such frames may vary, ignoring them and continuing in associated state, or making repeated attempts at re-authentication, or loss of connectivity.

3. When the client station probes for a particular network, the answering Probe Response frames are not expected to have certain FC flags set and are expected to contain valid ESSID and Supported Rates information elements. Based on the results of processing a response frame, the station may choose to proceed to authentication or ignore the responding network. The conditions under which the latter happens may differ between implementations.

4. The number of retransmits a station would attempt before giving up varies between implementations. As a convenient example, we consider the number of retransmissions of a purposefully unacknowledged Authentication Request once the station has been lured into attempting authentication. For example, we observed 14 retransmit attempts for the iPhone, 10 for a MacBook Pro, around 126–127 for the Intel PRO/Wireless 3945ABG operating under an open source Linux driver, and 3 for a Cisco 7920 wireless VOIP handset.

   Although some drivers allow changing retransmit counts, portable 802.11 devices typically do not expose it for easy changes (such as the Cisco 7920).

5. When the Authentication–Association 4-way handshake fails before completion, the reaction of the client station may vary; in particular, the number of attempts to establish the association. For instance, we observed the MacBook Pro sending apparently unlimited Probe Requests even after notifying the user of a connection failure, until the interface was manually brought down or the user manually selected another network. The iPhone, on the other hand, did not exhibit this behavior.

6. Presence or absence of beacon frames affects stations' association behavior in different ways. For example, we observed that the iPhone would not attempt to connect to an AP that answered its broadcast Probe Request unless it also sees beacon frames from that AP, whereas for the MacBook Pro the absence of beacons make no difference.[5]

In section 6 we describe our initial findings regarding platform behavior in these scenarios.

[5] In [5], the authors observed that some platforms would ignore unicast Probe Responses unless they also received beacons from the responding BSSID.

Chapter 4

Experimental setup

This chapter describes our experimental setup for testing the behavior of wireless stations. We also discuss some technical difficulties we encountered and ways to alleviate them.

4.1 Scanning and monitor platforms

Our testing system consists of an *scanning platform* and a *monitor platform* that run the respective testing and monitoring processes. These processes can share the same machine and even the same wireless interface, but preferably they should be physically separate for performance reasons.

The scanning platform is responsible for administering the stimuli. For single stage stimulus-response scenarios that do not require previous interaction with the device being fingerprinted, the scanning platform prepares and sends a series of stimulus frames. When possible, a MAC address in the frame identifies the particular test, e.g., malformed Probe Requests or Probe Responses are sent from non-existing stations and APs respectively. This allows us to speed up and "parallelize" testing.[1] The monitor platform sniffs the responses (which, when possible, identify the corresponding stimulus by the crafted MAC address) and converts the frame captures to the format suitable for the tests in the decision tree. For training, processing the results of single-stage tests can also be done offline, using the resulting packet capture file.

For multi-stage stimulus-response scenarios, in which the actual stimulus frame comes after an exchange intended to put the target into the state where the stimulus can elicit the desired response, the scanning platform is also responsible for playing the right sequence of prerequisite frames, emulating a normal station or an AP. Due to timing issues, only basic functions can be emulated reliably on commodity hardware.

We used the Atheros-based Ubiquity 802.11a/b/g card with the (modified) *madwifi-ng* driver for both the scanning and the monitor platforms, with the LORCON injection framework on the scanning platform, and the RFMON mode *libpcap*-based sniffing on the monitor platform.

4.2 Filtering and performance issues

When operating our monitoring platform in the lab, we experienced significant performance issues due to the high population of our wireless network. To reduce packet loss we found it necessary to delegate parts of the tests to the libpcap library's BPF filter and thus speed up the more costly tests performed on the captured frames above the *libpcap* layer. Unfortunately, the BPF filtering language does not include support for elements of the 802.11 protocol (although it accepts the *wlan* keyword, it is an alias for *ether*).

[1] A similar technique is used by port scanners to avoid the overhead of creating and maintaining connection data.

It does, however, support bitwise operations and tests on bytes and words at specified offsets and Boolean expressions on such tests, which it compiles into bytecode for the increased efficiency that we needed.

Thus we designed a simple filtering language exposing a set of 802.11 MAC header features similar to that of *Wireshark*'s display filtering language and implemented a generator of BPF filter expressions accepted by *libpcap*. Its output can be used with *libpcap*'s function `pcap_compile` or from the shell command line *tcpdump -F filter_file*, or directly from our Ruby BAFFLE classes:

```
PacketSet.new(Dot11, :type => 2).to_filter
> > > "(wlan[0] & 0xc) >> 2 = 2"
PacketSet.new(Dot11, :type => 0,
      :subtype => 11).to_filter
> > > "(wlan[0] & 0xc) >> 2 = 0 and
       (wlan[0] & 0xf0) >> 4 = 11"
```

The improvement in performance allowed us to conduct our experiments on our busy (40–90 APs on a typical *Kismet* scan) network.

4.3 Iterating over the stimuli sets

When testing devices for stimuli that could distinguish them from each other, we iterated over several subsets of the non-standard and malformed frames. As described above, we started with a subset of typically unused combinations of frame fields, represented as a subset of the Cartesian product of all possible combinations of chosen feature values (cf. Fig. 7.1).

Thus a typical test was specified by the frame type and subtype, the fixed elements of the frame, and ranges of values of varying elements. The resulting series of stimuli frames was represented by a Ruby object, which exported an iterator, which, when possible, also varied the MAC address encoding the identity of the stimulus. This approach, natural for the Ruby style of programming, made for ease of programming such series of tests. We expect it to be even more useful for further development of multi-stage tests, where iterators can be combined to quickly create new tests.

Chapter 5

Processing of responses

Once the target device has been subjected to all stimuli and its responses have been recorded, further processing takes place to present the results to the user. Besides providing the user with an at-a-glance visualization of the results (described later in 6), we answer the following questions:

1. What known device is the closest in its responses to the fingerprinted device, and how close is the resemblance?

2. Is the fingerprinted device one of the known trusted architecture(s)?

We use two methods to answer these questions. The former is answered by clustering, using the Singular Vector Decomposition (SVD), the latter by learning the description of the trusted set via Decision List learning.

5.1 Singular vector decomposition

We take the 8-bit flag field and run probes against each of its 256 combinations of flags five times. We track for which combinations of flags we get responses, and store the information in a 256-dimensional vector, so that each component ranges from 0 (never saw a response for this flag combination considered as an 8-bit integer) to 5 (always got a response for the flag combination). For each of the n devices being fingerprinted, we build these 256-dimensional vectors a fixed (k) number of times, and concatenate the feature vectors up into columns of a matrix that will have dimensions $256 * (n * k)$, recording which columns map to which device.

We then perform SVD on this matrix, which decomposes it into a product of three matrices. Then, we decide how many dimensions to trim the result matrices to (3 seemed to work nicely – you don't want too many as that would overfit, or two few, as that would lose too much information), and use the first two components of the decomposition to project each feature vector into a simpler 3-dimensional space. Now, every time we run a scan, we build the same 256-dimensional feature vector for the device being scanned, use the precomputed decomposition to project the new features into 3D, and find what known features are closest to the measurement, using a cosine distance metric.

For now, this is done individually for both types of flag fields we test for. It could probably be improved significantly by making those two flag fields a single 512-dimensional vector and training on that.

5.2 Automated decision tree derivation

Although currently our database of test results is relatively small, and decision trees for device signature tests can be easily derived by hand, we expect it to grow significantly, reaching the point where signatures will need to be derived in an automated way. We thus adapted a decision-list learning algorithm [12] with several heuristics to minimize the depth of the resulting lists.

Decision Lists (DL) are a representation of Boolean functions, efficiently learnable from examples and generalizing k-CNF and k-DNF formulae. A *k-decision list* consists of several consecutive nodes, each of which contains a conjunction clause of size no more than k. We use results of individual tests as input Boolean variables. Each test outcome represents a positive or negative example; the learned decision list for each known target platform represents a Boolean formula for recognizing that platform.

Our choice of decision-list learning is partly due to the intuition that, compared to other popular techniques, it tends to produce results that are typically more human-readable (and thus easier to manually tweak if needed).

Thus if a network administrator mandates a certain architecture for its stations or APs, the corresponding formula would combine the test results to answer whether the tested platform is of the mandated type. If other types are of no concern to the defender, checking that Boolean formula alone is sufficient for him (cf. our defensive scenarios in 1.0.3).

The algorithm described in [12] builds a decision list that represents a Boolean formula to correctly classify the majority of given examples (positive and negative). It does so by using an iterative term-by-term process. The errors accumulated because of preceding choices are partially corrected by each next term choice. The algorithm thus produces a series of partial solutions to the classification problem.

The original algorithm selects each next DL term at random. To shorten the depth of the resulting list, we select each next term so that the partial formula classifies the largest possible number of remaining examples[1]. We keep the pool of several candidates choices for each step, enabling limited backtracking. The resulting decision list is not guaranteed to be the shortest one, but our heuristics tend to shorten it significantly.

[1] This heuristic agrees with the visualizations of test results discussed in section 6.

Chapter 6

Experimental results

We used several scenarios from Sections 3.2 and 3.3 against a number of APs and client stations. We included two software-based APs, one based on the Atheros a/b/g card and the *madwifi-ng* driver, and the other based on the Prism2.5 card and the *hostap* driver. The choice of these popular software solutions was motivated, in part, by the likelihood of their attack use in “evil twin” scenarios (e.g., with tools like *Karma*).

6.1 Access Point fingerprinting

Our access points showed reproducible patterns of responses, confirming our basic hypothesis. We show a visualization of two series of our result tests in figures 6.1–6.5 and 6.6–6.10. All APs were configured for open access in default configuration. The first series *(ProbeFCTest)* tested responses to Probe Requests with varying FC flags, the second series *(AuthFCTest)* tested responses to Authentication Requests with similarly varying flags. We sent stimulus frames in random order to avoid any patterns due to timing and sequence of individual stimuli. Tests were repeated several times to minimize the effects of frame loss due to transient network congestion or other factors.

We visualize the results as follows. The 8 rows in each column represent the 8 Frame Control (FC) bits, descending from high-order bit to low-order bit. Bit cells are grey if set and black if unset. Each column represents one of the 256 combinations of bits we tested, and the width of a given column corresponds[1] to the relative frequency of responses to those bits in the test series.

Less frequent columns are harder to see and use up less horizontal space to avoid cluttering the display. Columns also fade out as their frequency decreases, to emphasize the importance of the high-frequency columns.

Immediately obvious in these visualization are the bit combinations (columns) which led to responses in every test. One can quickly gauge which bits were set in each of those columns. Our decision list learning algorithm is designed to favor the features corresponding to these bits.

These visualizations make apparent the differences in behaviors of the tested APs, except for the Linux-based Atheros and *madwifi-ng* soft AP and the Aruba AP70 with OpenWRT firmware, which exhibit similar patterns.

We are working on resolving timing and frame-loss sensitivity issues to produce reliable fingerprinting patterns in other scenarios.

[1] More precisely, column width is proportional to the fourth power of the relative frequency.

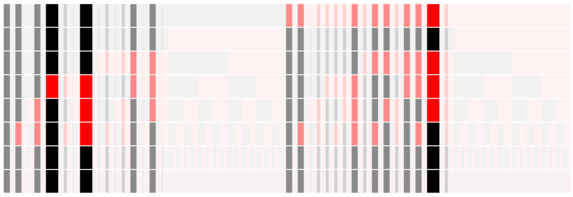

Figure 6.1: Cisco-Linksys WRT54g *AuthFCTest*

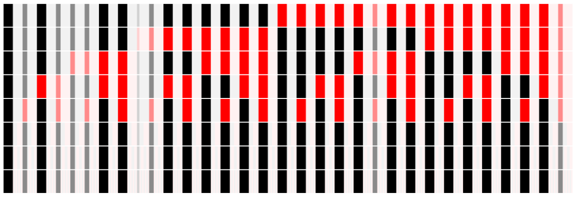

Figure 6.2: Extrasys WAP-257 *AuthFCTest*

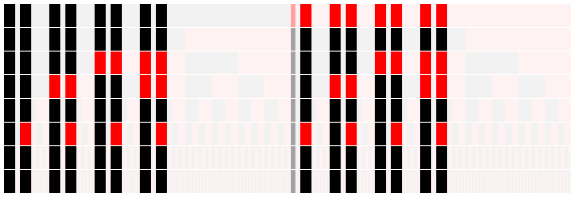

Figure 6.3: Madwifi-ng soft AP *AuthFCTest*

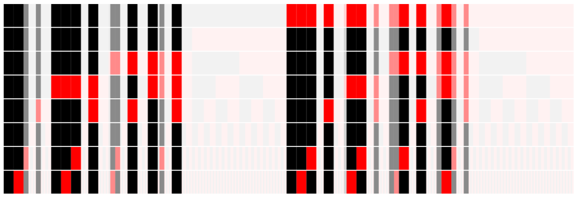

Figure 6.4: HostAP soft AP *AuthFCTest*

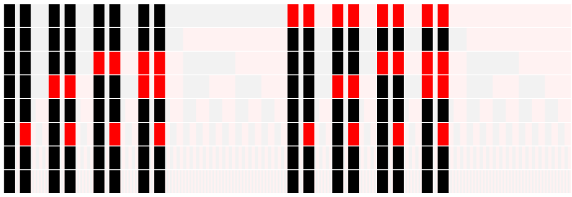

Figure 6.5: Aruba OpenWRT *AuthFCTest*

6.2 Station fingerprinting

Our station fingerprinting is a work in progress. Preliminary tests of scenarios from Section 3.3 with popular laptop brands (e.g., MacBook Pro and those with Intel Centrino chipsets) and portable devices (including an iPhone and a Cisco 7920) showed that client stations tend to ignore deauthentication and disassociation frame bits and also tend to attempt reconnection regardless of the Reason Code in these frames. We observed differences in the numbers of reconnection attempts and also in variation in timing of these attempts for different Reason Codes; however, more experiments are needed to confirm this behavior. Using Probe Responses as stimuli, however, produces a promising difference in behavioral patterns.

At present, we cannot realistically estimate false positive and false negative rates of our approach, for several reasons. Its active nature requires that we either own tested devices or obtain explicit consent from owners. As a result, our test equipment set is rather small. Following the example of *nmap* and other classical tools in providing an open fingerprinting network and inviting community contributions, we expect to significantly increase the recognized device set. We note that since the actual set of deployed devices is a fast moving target, providing such a framework is the only hope that a free fingerprinting tool has of retaining its usefulness.

6.3 Incidental observations

In the course of our experiments we encountered 802.11 implementation behaviors that, although not described by any standard, could, in combination with other features of their respective platforms, have unexpected security implications.

Association behavior of a client station after introduction of a rogue AP with a stronger signal provides several examples. The Shmoo group observed versions of Windows XP prior to SP1 even leaving the selected ESSID and associating with a stronger signal AP without warning the user [8]. We observed similar behavior in Orinoco Gold cards under an older version of a Linux *orinoco_cs* driver.

We encountered an even more interesting behavior in the Linux driver for the Intel 4965 a/b/g interface: in the presence of a stronger 802.11a AP advertising the same ESSID, it would abandon its working 802.11g association and associate with the 802.11a AP instead (unless explicitly limited to the b/g mode with an *iwpriv* call). In an environment where 802.11b/g range is monitored by a WIDS, whereas 802.11a is not normally used or monitored, such switching may facilitate a successful "under the radar" MITM attack. We believe that *classifying and cataloging* such behaviors should be a part of further behavioral fingerprinting.

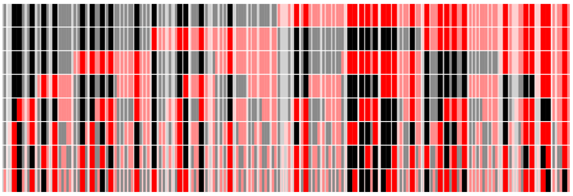

Figure 6.6: Cisco-Linksys WRT54g *ProbeFCTest*

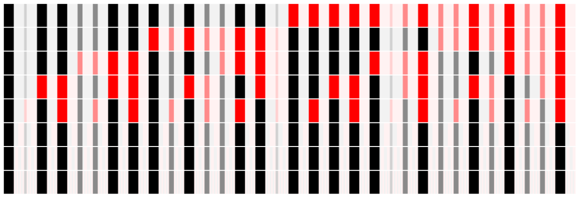

Figure 6.7: Extrasys WAP-257 *ProbeFCTest*

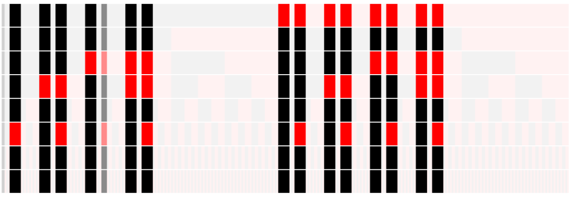

Figure 6.8: Madwifi-ng soft AP *ProbeFCTest*

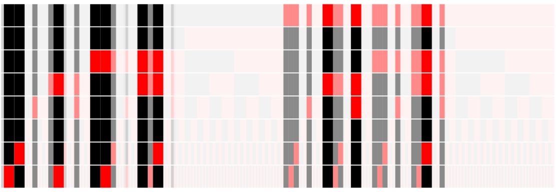

Figure 6.9: HostAP soft AP *ProbeFCTest*

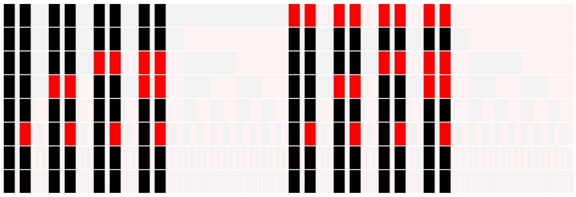

Figure 6.10: Aruba OpenWRT *ProbeFCTest*

Chapter 7

Conclusion

We tested a number of scenarios for active fingerprinting of 802.11 MAC layer implementations that rely on exchanging crafted 802.11 frames with the fingerprinted device. Our approach complements other fingerprinting approaches, and can yield faster results, for defensive uses in particular. Our results show this approach to be feasible for access points, and also suggest promise in fingerprinting client stations. We also present a visualization of our sample test results and outline several directions for future investigation.

Acknowledgements

We are grateful to Joshua Wright, Michael Kershaw, and John Ellch for getting us interested in 802.11 fingerprinting and releasing many useful tools without which this project would not be possible.

This research program is supported in part by Grant 2005-DD-BX-1091 by the Bureau of Justice Assistance, a component of the Office of Justice Programs, which also includes the Bureau of Justice Statistics, the National Institute of Justice, the Office of Juvenile Justice and Delinquency Prevention, and the Office for Victims of Crime. Points of view or opinions in this document are those of the authors and do not represent the official position or policies of the United States Department of Justice.

Bibliography

[1] PROTOS – security testing of protocol implementations. http://www.ee.oulu.fi/research/ouspg/protos/.

[2] Laurent Butti. Wi-Fi advanced fuzzing. Black Hat Europe, February 2007.

[3] Johnny Cache. Fingerprinting 802.11 implementations via statistical analysis of the duration field. *Uninformed.org*, 5, September 2006.

[4] Johnny Cache and David Maynor. Hijacking a MacBook in 60 seconds. Black Hat, August 2006.

[5] Johnny Cache, H D Moore, and skape. Exploiting 802.11 wireless driver vulnerabilities on Windows. *Uninformed.org*, 6, January 2007.

[6] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of 15th USENIX Security Symposium*, pages 167–178. USENIX, August 2006.

[7] K. N. Gopinath, Pravin Bhagwat, and K. Gopinath. An empirical analysis of heterogeneity in IEEE 802.11 MAC protocol implementations and its implications. In *WiNTECH '06: Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, pages 80–87, New York, NY, USA, 2006. ACM Press.

[8] The Shmoo group. 802.11 bait, the tackle for wireless phishing. Toorcon, San Diego, October 2005.

[9] M. Laakso and M. Varpiola. Vulnerabilities go mobile. In *AusCERT Asia Pacific Information Technology Security Conference*, May 2002.

[10] Niels Provos. A virtual honeypot framework. In *Proceedings of 13th USENIX Security Symposium*, pages 1–14. USENIX, August 2004.

[11] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection, January 1998.

[12] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[13] Kathy Wang. Frustrating OS fingerprinting with Morph. DEFCON 12, August 2004.

		Order		Protected Frame		More Data		Power Management		Retry		More Fragments		From DS		To DS	
		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Management	Association Request				●	●	●	●	●	●	●	🟣	●		●		●
	Association Response				●	●	●	●	●	●	●	🟣	●		●		●
	Reassociation Request				●	●	●	●	●	●	●	🟣	●		●		●
	Reassociation Response				●	●	●	●	●	●	●	🟣	●		●		●
	Probe Request	🟥	🟥	🟨	🟨 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 🟣	🟥 ●	🟥	🟥 ●	🟥	🟥 ●
	Probe Response	🟥	🟥	🟥	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 🟣	🟥 ●	🟥	🟥 ●	🟥	🟥 ●
	Beacon				●	●	●	◦	◦	◦	◦	◦	◦		◦		◦
	ATIM				●	●	●	●	●	●	●	🟣	●		●		●
	Disassociation				●	●	●	●	●	●	●	🟣	●		●		●
	Authentication	🟥	🟥	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 ●	🟥 🟣	🟥 ●	🟥	🟥 ●	🟥	🟥 ●
	Deauthentication				●	●	●	●	●	●	●	🟣	●		●		●
	Action				●	●	●	●	●	●	●	🟣	●		●		●
	4 * Reserved																
Control	BlockAckReq												●		●		●
	BlockAck												●		●		●
	Power Save (PS)-Poll												●		●		●
	Request To Send (RTS)												●		●		●
	Clear To Send (CTS)												●		●		●
	Acknowledgment (ACK)												●		●		●
	Contention Free (CF)-End												●		●		●
	CF-End + CF-ACK												●		●		●
	8 * Reserved																
Data	Data	●	●		●					●	●	●	●	●	●	●	●
	*Data + CF-ACK																
	*Data + CF-Poll																
	*Data + CF-ACK + CF-Poll																
	Null Function (no data)	●	●		●					●	●	●	●	●	●	●	●
	*CF-ACK (no data)																
	*CF-Poll (no data)																
	*CF-ACK + CF-Poll (no data)																
	*QoS Data																
	*QoS Data + CF-ACK																
	*QoS Data + CF-Poll																
	Qos Data + CF-ACK + CF-Poll																
	*QoS Null (no data)																
	*QoS CF-Poll (no data)																
	*QoS CF-Poll + CF-ACK (no data)																
	Reserved																
Reserved																	

Legend

- ● Defined by IEEE 802.11 Specification
- ◦ In IEEE 802.11 Specification but purpose seems undefined
- 🟣 In IEEE 802.11 Specification but unlikely
- 🟥 Tested by BAFFLE
- 🟨 Tested by BAFFLE but of limited utility
- □ Not defined in IEEE 802.11 Specification
- * In IEEE 802.11 Specification but mostly unimplemented

Table 7.1: Allowed and explored combinations of flags and frame types