



Improvement of 802.11 fingerprint diversity

Clémentine Maurice

► **To cite this version:**

Clémentine Maurice. Improvement of 802.11 fingerprint diversity. Cryptography and Security [cs.CR]. 2012. dumas-00730426

HAL Id: dumas-00730426

<https://dumas.ccsd.cnrs.fr/dumas-00730426>

Submitted on 10 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITY OF RENNES 1 – INSA RENNES
RESEARCH MASTER'S DEGREE IN COMPUTER SCIENCE

Improvement of 802.11 fingerprint diversity

Internship report

Written by
Clémentine Maurice

Supervised by
Olivier HEEN
Christoph NEUMANN
Stéphane ONNO

Security & Content Protection Lab
Technicolor, Rennes

Abstract 802.11 networks are largely adopted, thus the identification of wireless devices becomes a major issue in network security. This study takes place in the scope of a defensive scenario, and can be used to detect Medium Access Control (MAC) address spoofing. We develop an approach to improve the identification of unique devices while keeping existing 802.11 fingerprinting methods. We evaluate our approach on two state-of-the-art methods and successfully improve the identification of identical devices on one method.

June 5, 2012

Acknowledgments

First, I would like to particularly thank my internship supervisors Olivier Heen, Christoph Neumann and Stéphane Onno for allowing me to discover and work on this topic. Their commitment, guidance and expertise made this internship very pleasant and interesting.

I also would like to thank the Security and Content Protection Lab of Technicolor and the other interns for their warm welcome and cheerfulness. Technicolor and the team have made this internship to take place in good conditions.

I finally thank the INSA Rennes and the University of Rennes 1 for their pedagogic and administrative support.

Contents

Introduction	3
1 802.11 fingerprinting	4
1.1 802.11 security: the need for non-cryptographic techniques	4
1.2 Overview of 802.11 fingerprinting	5
1.3 Fingerprinting drivers	7
1.4 Fingerprinting network interface cards	8
1.5 Fingerprinting unique devices	8
1.6 Fingerprinting users	10
2 Diversity in fingerprinting	11
2.1 Improvement of fingerprinting and motivations	11
2.2 The “diversity” approach	12
2.3 Discussion	13
2.4 Evaluation	13
3 Diversity in Cache’s method	16
3.1 Background: carrier sensing in 802.11	16
3.2 Cache’s method	16
3.3 Cache’s method without diversity	18
3.4 Cache’s method with diversity	19
3.5 Future work	23
3.6 Conclusion	25
4 Diversity in Franklin et al. method	26
4.1 Background: active scanning in 802.11	26
4.2 Franklin et al. method	27
4.3 Franklin et al. method without diversity	28
4.4 Franklin et al. method with diversity	28
4.5 Future work	31
4.6 Conclusion	32
5 Future work	33
Conclusion	34
A Dataset	35
B Driver architecture and code	38
C Active scanning’s simulation code	40
D Packet injection’s code	42
Bibliography	44

Introduction

802.11 networks – which are often referred to as “Wi-Fi networks” – are largely adopted thanks to the standardization, the interoperability and the low cost of wireless-enabled devices. With the profusion of these devices comes the need to secure wireless networks. One major issue is then the identification of wireless devices. The IEEE 802.11 standard provides some cryptographic techniques for this purpose, that turn out to be cryptographically weak for the Wired Equivalent Privacy (WEP) – that is now deprecated – and not sufficient for the Wi-Fi Protected Access (WPA) as not all exchanged frames are protected. Some non-cryptographic techniques emerge on the sidelines of the standard to overcome its limits. In this study we focus on fingerprinting, that is the action of identifying a device by extracting some externally observable characteristics. 802.11 fingerprinting is actually a young field of research that is less than ten years and, like the intrusion detection field, suffers from false alarms that lead users to ignore them.

In this internship, we develop an approach called *diversity* that improves unique devices identification. Our intention is to keep the existing fingerprinting methods, using them unmodified as black boxes, and to slightly modify some observed attributes themselves in a random way to differentiate the devices. We work within the scope of home network security and we state the hypothesis that the fingerprinter is the “good guy” and has a certain control over devices. We tested and evaluated the diversity approach on two state-of-the-art fingerprinting methods. For one fingerprinting method we successfully improve the identification of unique devices.

The remainder of the report is organized as follows. Chapter 1 describes the state-of-the-art 802.11 fingerprinting methods. Chapter 2 develops the diversity approach. Chapter 3 and 4 are dedicated to the instantiation of the approach on two existing fingerprinting methods. Chapter 5 indicates possible directions for future work and is followed by a conclusion of the study.

Chapter 1

802.11 fingerprinting

In this chapter, we are interested in depicting 802.11 fingerprinting. We first summarize the security of 802.11, and establish the need for other security techniques such as fingerprinting. Then we give an overview of the general fingerprinting approach, and we finally summarize state-of-the-art methods.

1.1 802.11 security: the need for non-cryptographic techniques

802.11 is a set of standards ratified by the IEEE, describing characteristics of a Wireless Local Area Network (WLAN) – which is often referred to as “Wi-Fi network”. The adoption of WLANs has increased recently due to this standardization and the interoperability between wireless devices. Yet, the broadcast nature of wireless networks – and in particular 802.11 networks – makes security a challenge, compared to wired networks: communication can easily be eavesdropped or intercepted. Moreover, the identification of users and devices solely relies on Media Access Control (MAC) address of Network Interface Cards (NICs), and the access control is generally enforced by these addresses, which makes it vulnerable to identity-based attacks created by MAC address spoofing.

Some cryptographic techniques can be used to provide confidentiality of data frames. The Wired Equivalent Privacy (WEP) protocol brought by the original standard [1] in 1997 has shown to be cryptographically weak [13]. Wi-Fi Protected Access (WPA) has been used as a short-time fix to replace the deprecated WEP in 2003, waiting for the complete 802.11i standard [2]. WPA introduces improved authentication mechanisms reusing the legacy hardware. In 2004, the 802.11i standard – referred to as Wi-Fi Protected Access 2 (WPA2) – is ratified. It provides effective data confidentiality and integrity at the MAC layer [19]. We differentiate WPA-personal that operates with a pre-shared key, and WPA-enterprise that requires an authentication server. However, 802.11i assumes an upgrade of the hardware, and does not protect management and control frames that remain unauthenticated and unencrypted to provide retro-compatibility. The mutual authentication between stations and access points has also shown some weaknesses, and does not completely prevent rogue access points [36]. It comes that even with the additional security mechanisms, identity-based attacks are still feasible and they generally are the first step to serious attacks such as man-in-the-middle (MITM) or session hijacking.

To protect 802.11 networks from MAC address spoofing without cryptography, various techniques have been identified by Samra et al. [30] and Zeng et al. [35]. They both cite control of sequence numbers¹, location determination and fingerprinting. Samra et al. also cite Signal

¹Some Wireless Intrusion Detection Systems (WIDS) like Snort Wireless (<http://snort-wireless.org/>)

Strength Fourier Analysis that measures interferences between the attacker and the victim.

1.2 Overview of 802.11 fingerprinting

In this study we focus on 802.11 fingerprinting and we refer to the *fingerprinter* as the device performing fingerprinting, and the *fingerprinttee* as the device being fingerprinted.

1.2.1 The applications

Fingerprinting is the identification of 802.11 devices (station or access point) by its externally observable characteristics. It results in a signature – an identifier for the device being fingerprinted – and a classification of the device. The observed characteristics may concern the physical layer as reported by Danev et al. in [10] (radio communication, clock, network card or chipset) as well as the MAC layer. The complexity of the 802.11 standard has led to variety in implementations, whether due to failure to comply with the standard, or lacks in the standard itself. The heterogeneity of behaviors enables the process of fingerprinting, as shown by Gopinath et al. in [16]. The field is quite young, as the first publications on the subject date from 2004-2006, for a standard normalization in 1997.

Fingerprinting can be used in a defensive way, to detect and to prevent MAC address spoofing for stations and rogue access points. These are situations that the current Intrusion Detection Systems (IDS) fail to detect and that should be detected by a good fingerprinting method. Sieka proposes in [32] to use it to detect impersonation – where a node pretends to be an other node – and sybil attacks – where a node assumes multiple identities. Fingerprinting can be used in an offensive way as well. The ability to identify a configuration can be employed to target a specific vulnerability and launch a driver-specific exploit. The attacks are reinforced by the general poor quality of driver code, the execution in kernel space and the easy remote interaction, as stressed by [14].

Generally speaking, fingerprinting is useful in situations where there is no specific context of trust between machines, for instance a shared secret key. It assumes no collaboration between the fingerprinter and the fingerprinttee. To build trust, the machines need to be sure of the identity of the ones they communicate with. Fingerprinting can then be implemented as an additional layer of trust, before or after a key-based authentication.

Nevertheless, fingerprinting raises the question of privacy. One way to ensure anonymity in 802.11 networks is to change regularly its MAC address. Identifying devices with its observable characteristics could be used as a mean to track devices or even users. On the contrary, Lackner et al. mention in [25] that understanding the principles of fingerprinting is also a way to prevent it.

1.2.2 The general method

What do we fingerprint? Cache [6] points out that “an implementation comprises a driver, radio chipset, firmware, and possibly some user-space applications”. State-of-the-art methods often look at one attribute to identify the devices – though Idland [20] proposes a method that combines several attributes. Two methods that don’t identify the same “objects” are not comparable. As Danev et al. mention in [10], finding the causes of the unique identification of devices – that is seeking the exact “object” fingerprinted – is a difficult task but is crucial in terms of defensive and offensive usages. It still is an open issue on some existing methods.

already implement the control of sequence numbers.

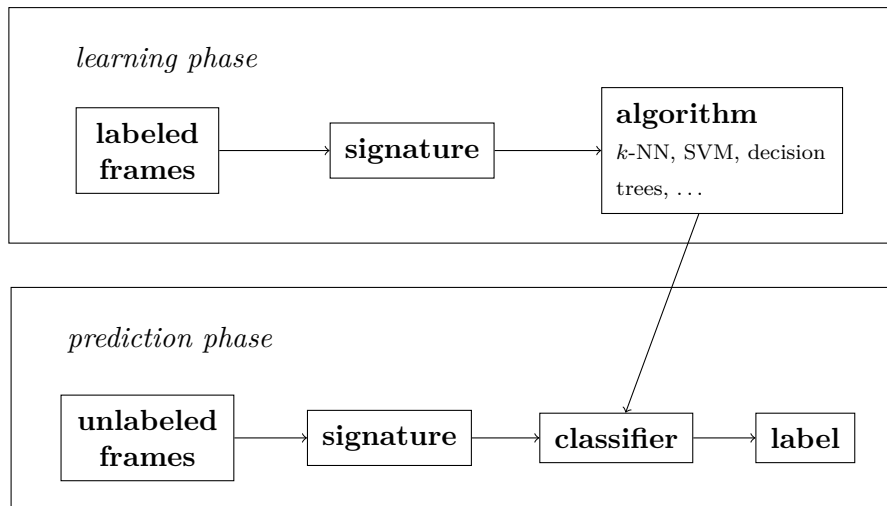


Figure 1.1: Fingerprinting process.

How do we fingerprint? Two different fingerprint categories exist: passive and active methods. Passive methods just observe packets from the network and analyze them; they are stealth and difficult to prevent. In active methods, the fingerprinter needs to transmit data to the fingerprintee and observes subsequent traffic; they are detectable and possibly harmful to the fingerprintee. Fingerprinting can be done using a standard wireless card or several cards (e.g. one card per channel to observe). Physical fingerprinting methods sometimes use software-defined radio or oscilloscope [10]. We are interested in methods using standard cards.

How do we classify? Fingerprinting is basically supervised learning. The process is shown in Figure 1.1 and follows these steps:

- The first step of the **learning phase** is to gather the training set, that are wireless traces with the devices' *labels* (*i.e.* we know what are the different devices). It can be achieved by monitoring some wireless network with a simple 802.11 card in monitoring mode and the `pcap` library², or with publicly available traces, such as traces from Sigcomm conferences³. Then we extract from the traces the relevant attributes, that will distinguish devices from each other. The next step is to apply the learning algorithm [34] to the training set: decision trees as in [4], Bayesian classifiers as in [29], Support Vector Machine (SVM) as in [31] or nearest neighbors as in [27].
- In the **prediction phase** we have unlabeled traffic from a device that we want to identify. The first steps are the same than the learning phase: we gather the traffic and extract the relevant features to build the unknown device's signature. The classifier then compares this signature to a base of known devices' signatures built in the learning phase.
- Finally, during the **evaluation** of the method, we run the learning algorithm on a test set – separated from the training set. Some metrics can assess the validity, such as the *True Positive Rate (TPR)* which is the fraction of the test set that has been correctly identified. We also use recall and precision, metrics that we borrow from the Information Retrieval domain. Section 2.4 is devoted to the details of the evaluation process.

²<http://www.tcpdump.org/pcap.html>

³<http://crawdad.cs.dartmouth.edu/meta.php?name=uw/sigcomm2004>

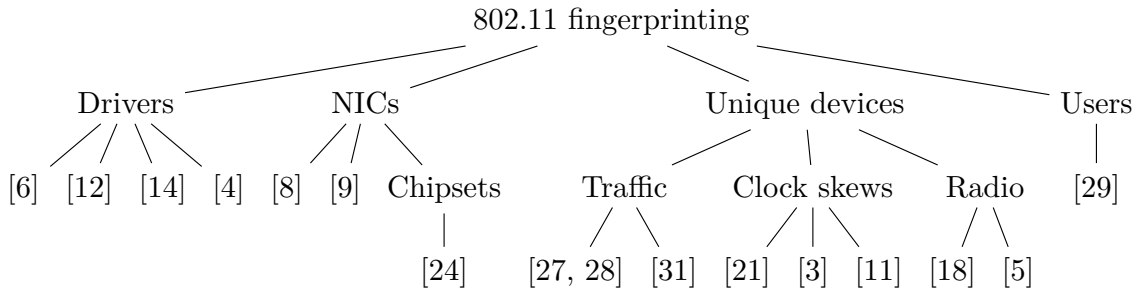


Figure 1.2: Fingerprinting map.

1.2.3 The different methods

We choose to classify the 802.11 fingerprinting methods according to the “objects” fingerprinted: drivers, Network Interface Cards (NICs) and chipsets, unique devices and finally users. A map representing the classification is given in Figure 1.2. It should be noted that few papers give hints about what their method actually classifies. For example, Corbett et al. declare in [8] that their method identifies NICs using active scanning whereas this procedure is typically implemented in software, and thus in drivers. Our classification relies on papers claims.

1.3 Fingerprinting drivers

Cache – Duration field [6] Cache developed a passive method that specifically fingerprints drivers, using the duration field present in most of the frames. The duration field is a 16 bit value which describes how long (in microseconds) a station intends to keep the medium access after the current transmission, including replies from the recipient. The different drivers don’t calculate this field in the same way, leading to variations. We use this method to test the diversity approach. More details are given in Chapter 3.

Cache – Association redirection [12] Cache also proposed an active method that fingerprints drivers. To join a network, a station must first authenticate to the access point with an authentication request and wait for the access point to send authentication response; then it must associate with an association request and wait for an association response. The fingerprinting method uses the process of association-redirection, where an access point sends the association response with an other address than the one in the authentication response. The authentication and association processes are implemented in wireless drivers. Cache found nine unique situations, depending on the stations’ responses.

Franklin et al. – Active scanning [14] Franklin et al. established a passive fingerprinting method that uses active scanning to fingerprint drivers. When a station seeks a wireless network to join, it can use passive or active scanning. In active scanning, the station sends probe requests and waits for probe responses from access points. The method is based on the time between consecutive probe requests in a single channel. It measures the time between probe requests in a burst in the same channel, as well as the time needed to cycle through the different channels. We also use this method to test the diversity approach. More details are given in Chapter 4.

Bratus et al. – Responses to malformed frames [4] Bratus et al. developed a method based on the heterogeneity of 802.11 MAC implementation. This method is active and can

fingerprint stations and access points. The hypothesis is that various implementations lead to various reactions to non-standard events. The authors charted the possible combinations of features derived from the fields of the 802.11 MAC headers, and especially the 8 last bits of the Frame Control subfield (thus 256 combinations). They marked non-standard or unusual combinations. The fingerprinting method is then based on stimulus-response. Sending a stimulus and waiting during a timeout period, they consider three types of responses: any frame transmitted, a specific type and subtype of frame transmitted, or no frame transmitted. They experimentally show that they can differentiate between several devices but don't specifically determine if the differences in behavior result from the chipset, driver or firmware. As the MAC layer tend to be implemented in software, the method should fingerprint the driver.

1.4 Fingerprinting network interface cards

Corbett et al. – Active scanning [8] Corbett et al. developed a passive fingerprinting method that uses active scanning – like Frankin et al. [14] – to identify NICs. The process iterates through the different channels, creating a periodic behavior that the authors analyze with spectral analysis. They use the probe request frames and their associated timestamps to represent the time series of events. The experiment was conducted with three different cards, that exhibited a distinctive spectral profile. However, each card using a different driver, we suspect that the experiment can't conclude that it differentiates NICs more than drivers.

Corbett et al. – Rate switching [9] Corbett et al. proposed an other method that passively classifies NICs. It uses rate switching, an algorithm under-specified in the standard that adapts the transmission rate depending on the quality of the link. The algorithm impacts the traffic pattern: the inter-arrival time between frames can be shorter or longer. The authors use spectral analysis to study the periodicity caused by rate switching in the wireless traffic. They use the transmission rate and associated timestamps to represent the time series of events. Over six NICs, Corbett et al. differentiate three different behaviors. By comparing cards from different manufacturers and from the same manufacturer, they suggest that rate switching may be implemented in hardware.

Lackner et al. – Chipset's checksum verification [24] Lackner et al. determined a fingerprinting method that passively fingerprints the chipset. It is based on the time between the reception of a frame and the transmission of the acknowledgment. During this time, the chipset evaluates the received packet's checksum with a cyclic redundancy check algorithm, an error-detecting code implemented in hardware. The computing time depends on the implementation of the algorithm. The authors used self organizing maps for the training with seven different chipsets, and achieved 80% of accuracy.

1.5 Fingerprinting unique devices

1.5.1 Global traffic based

Sieka – Access point's authentication period [31] Sieka developed a method that actively fingerprints access points. It analyzes timings during the authentication procedure. In the authentication process, the station sends an authentication request to which the access point responds by an acknowledgment, closely followed by an authentication response. The method requires a station sending requests, as well as the fingerprinter – a monitoring station – and

the fingerprintee – an access point. The fingerprinter computes the time between the acknowledgment and the authentication response, and classify access points with SVM classifiers. The experiment setup is composed of five different access points, four of which have the same brand, model and chipset. Yet, the method is able to differentiate between them with an accuracy of 86%.

Neumann et al. – Inter-arrival time [27, 28] Neumann et al. established a method based on the frame inter-arrival time, the time between the end of receptions of two consecutive frames. They build an histogram with the frequencies of inter-arrival times for each device. The inter-arrival time comprises: (1) *the frame transmission duration*, that depends on a mix of wireless cards features and on the application generating the data, and (2) *the idle period*, that mainly depends on the wireless card and driver. The authors show that the inter-arrival times are thus characteristic of unique devices.

1.5.2 Clock skew based

Jana et al. – Access point’s clock skew [21] Jana et al. proposed a method to detect fake access points based on clock skews, a cumulative off-set from one device to the other. Kohno et al. have already shown in [23] that the clock skew is a per-machine constant that remains fairly consistent over time in a Wide Area Network (WAN). In the wireless side, Jana et al. calculate the clock skews of access points using the timestamps in beacons frames. These frames contain all the information about the network and are transmitted periodically by the access points. The approach needs very high precision timers to compute the minute deviation of the clock. The authors developed two methods that have a different tolerance towards outliers, depending on the desired false negative rate and security context.

Arackaparambil et al. – Access point’s clock skew [3] Arackaparambil et al. pursued the work of [21]. They found significant changes in the implementation of the timer used to calculate the clock skews. Since this measurement is critical to computation, they investigate a new method that would be more accurate, using a hardware-based clock and not software-based. Arackaparambil et al. also present an attack that changes the clock skew of a fake access point to imitate that of an authorized one. They use virtual interfaces – station and access point – that acquire the clock skew of the legitimate access point the virtual station is connected with. The fingerprinting method of Jana et al. couldn’t distinguish the two access points.

Desmond et al. – Active scanning + clock skew [11] Desmond et al. pursued the work of [14] that fingerprints drivers. They found minute differences in timing intervals between consecutive probe requests emitted by different machines with the same driver. They therefore generalize the approach of Franklin et al. to differentiate between unique combinations of the tuple {machine, NIC, driver, OS}. Their intuition is that the timings also depend on the machine’s operating system and clock skew, and some variability due to noise. To differentiate between two different machines running with the same NIC, driver and OS, they propose a method that separates noisy data from noiseless data.

1.5.3 Radio frequency fingerprinting

Another field in hardware fingerprinting is radio frequency fingerprinting. 802.11 devices are composed of radio transceiver to send and receive data. They also have unique characteristics, often based on minute hardware imperfections acquired at manufacture. Danev et al. [10]

report two categories of radio frequency fingerprinting: transient-based and modulation-based approaches. The transient-based approach extracts the transient portion of the signal, the brief radio emission prior to transmission where the amplifier goes from idle to the level required for data communication. This approach was developed by Hall et al. in [18]. Modulation-based approach extracts features from the part of the signal that has been modulated, *i.e.* the data. Brik et al. developed this approach in [5].

However, radiometric fingerprinting requires expensive signal analyzer to be performed, that is why we do not take into account this category in the remainder of this report.

1.6 Fingerprinting users

Pang et al. – Privacy vs. implicit identifiers [29] The issue addressed by Pang et al. is quite different from the others, taking the perspective of privacy. The 802.11 standard does not provide anonymity to prevent any location tracking based on the MAC address, transmitted with every frame. To mask this identifier, a common way to do is to apply pseudonyms by periodically change the MAC address of the 802.11 devices. However, they show that this is insufficient to provide anonymity in 802.11, because the traffic itself can identify the users implicitly. The attributes they focus on are: network destinations, SSID probes, broadcast packet sizes and MAC protocol fields. Such combination of attributes can be viewed as a signature.

Chapter 2

Diversity in fingerprinting

In this chapter, we first explain the necessity of an improvement of 802.11 fingerprinting. We then detail our generic approach and the way we conduct our research. Finally, we discuss the choices we make and the issues that are still open.

2.1 Improvement of fingerprinting and motivations

Danev et al. [10] describe the properties of a good physical fingerprint for wireless devices – 802.11 or not. These are still valid for 802.11 fingerprinting, whether it fingerprints the MAC layer or the physical layer:

- universality: each device we want to fingerprint must have the considered features;
- uniqueness: two different devices must have different signatures;
- permanence: the signatures must be invariant over time;
- collectability: we must be able to collect the features with available equipment;
- robustness: fingerprints must not be influenced by environmental factors, whether they are external (e.g. surrounding material) or internal (e.g. temperature).

While the robustness criterion is only cited for physical fingerprinting, it still has its importance for MAC layer fingerprinting that takes into account the environment, and more precisely the influence of other 802.11 devices.

Our goal is to improve uniqueness of existing fingerprinting methods. Improving fingerprinting is important to avoid the main issue of current IDS that is false alarms (false positives). False alarms, when they are too frequent, often lead users to ignore alerts, even when there is an actual intrusion. To improve the performances of 802.11 fingerprinting in a traditional fashion, at least three ways can be envisioned:

1. We can change the selection of attributes that will distinguish devices from each other (frame inter-arrival time, sizes of broadcast packets, ...) and find more discriminant attributes. Given the variety of attributes that has been found so far, this can be a difficult task.
2. We can change the learning algorithm: replace an algorithm by an other or change the parameters of the employed algorithm (for the nearest neighbors, the similarity measure). The inconvenient is that not all algorithms of supervised learning can be applied to all situations, and they do not have an infinity of parameters to adjust.

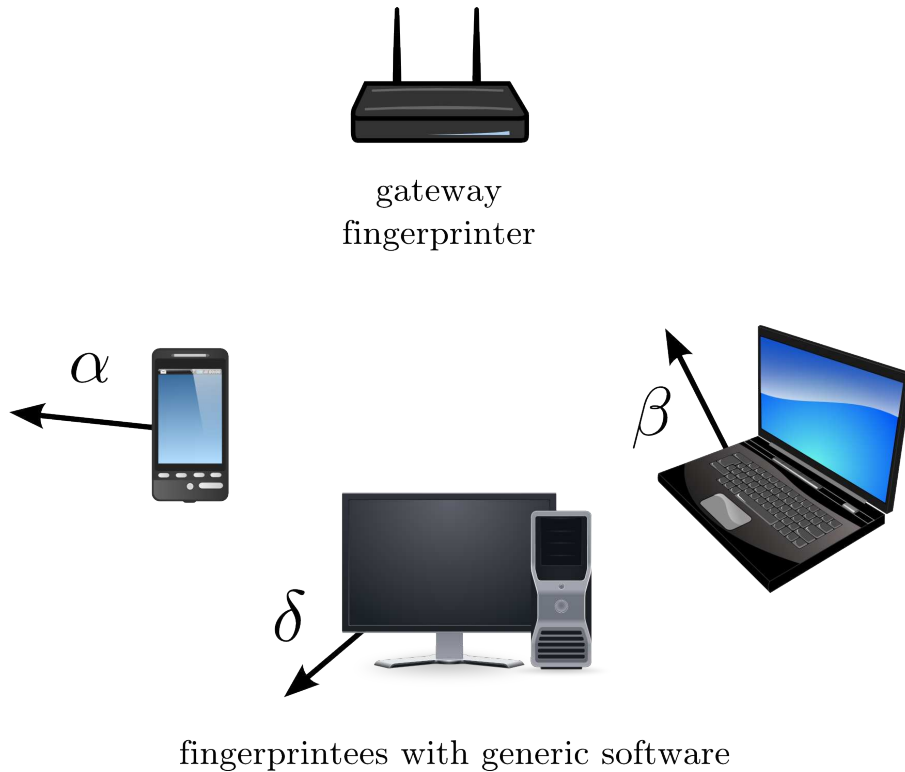


Figure 2.1: Example of setup for the diversity approach of 802.11 fingerprinting. The gateway is the fingerprinter and the fingerprintees are moved away from each other with modifications α , β or δ .

3. We can combine several fingerprinting methods. However, we saw in Section 1.2 that two methods don't necessarily identify the same configuration. For instance, one may identify drivers, whereas the other may identify chipsets. The combination of both methods won't result in an improvement if we still want to identify drivers.

Moreover, in the last case, the combination of heterogeneous methods may lead to undecided situations where e.g. the identification of a chipset is incompatible with the identification of the driver.

2.2 The “diversity” approach

Based on these observations, to be able to better differentiate between devices, we consider a new approach that we call *diversity*. Our intention is to keep the existing methods, using them unmodified as black boxes, and slightly modify the attributes of the fingerprinted devices themselves in a random way. The goal is to move the signatures calculated by the fingerprinter away from each other and have a statistically greater distance between them. We focus on the case where the fingerprinter is “the good guy” – fingerprinting being used in a defensive way. Thus we have a certain control over legitimate devices to be able to modify their attributes. However, there still is no cooperation between the fingerprinter and the fingerprintees: the change in the attributes is not related to any secret value. The hypothesis of the fingerprinter being “the good guy” has some validity in this context because Technicolor produces gateways for domestic networks. To provide security in these networks, the gateway can be the fingerprinter

and diversity can be implemented on devices with a generic software. A figurative example of the approach is given in Figure 2.1. We define diversity :

Definition 1 *For a fingerprinting method that uses some discriminating attribute having value x , the diversity modifies this value by $x + \alpha$ for a device, $x + \beta$ for an other, and $x + \delta$ for a third one. The fingerprinting method remains exactly the same: learning phase, prediction phase and evaluation.*

To test the diversity approach, we use the following methodology. First, we choose a fingerprinting method and study it in details. Then we explore the need for diversity by evaluating the original method in an unfavorable case – a dataset with devices having identical fingerprints. We next search for the attributes to modify, we simulate diversity and evaluate our simulation to see if there is any improvement. Finally, we change the actual behavior of devices if possible and evaluate the same method on the new dataset.

2.3 Discussion

To use the diversity approach in practice, we have to make choices. The first choice is about the fingerprinting method: the observed characteristics must be modifiable – else the approach don’t apply –, the fingerprinting method should be easily implementable and reproducible for tests. We also choose to focus on passive methods that are less invasive for the fingerprintees. When the method is selected, next choice is about the attribute to change. Its modification must produce a change in the device’s signature, but must not interfere with the normal behavior of the device, or even surrounding devices. For example, some timings that are part of the 802.11 standard should not be changed. Finally, when the attribute is selected, the last choice is about the intensity of the perturbation. Two signatures that were not differentiated by the learning algorithm must be different enough thanks to the diversity, but the normal behavior of the device should still not be disrupted.

The diversity approach also bring some issues. First, “Should the modifications be fixed once for all and by whom?” is a question that affects the privacy and the fingerprinting process. If we periodically change the modifications, the devices’ signature also changes and we have to deal with it in the learning phase. For the time being, we decide to make the modifications persistent. Another question is: “Should the modifications of one device depend on the modifications of the other fingerprintees?”. We can imagine that if the modifications are different from one device to an other, the approach will be more effective. Actually, it supposes that the devices can all monitor the neighborhood traffic and adapt their own traffic, which would require two network cards per device and is not a common setup. The “object” classified by the method is also to be considered with the intensity of perturbations: “If the method originally identifies drivers, do we still want it to identify drivers, or unique devices?”. In this study, we focus on identifying unique devices, the ability of the methods to still identify drivers is still an open issue. Finally even in a home network, some devices cannot be controlled or modified, for example connected televisions. The partial diversity approach still has an interest to differentiate the controllable devices with the uncontrollable ones, and thus globally improve the identification of all devices.

2.4 Evaluation

In this study, we choose to test the diversity approach on two fingerprinting methods: Cache [6] and Franklin et al. [14]. Cache’s method depends on durations’ histograms that could be modified in several ways (changing or adding values, changing ratios. . .); Franklin et al. method

	<i>mixed set</i>		<i>unique set</i>
method	Franklin	Cache	Franklin / Cache
configurations	19	17	9
different drivers	9	9	1
best to characterize	drivers		unique devices

Table 2.1: Characteristics of the two sets of devices.

relies on timings of active scanning that is implemented in software and is likely to be modified. They both fingerprint drivers: several devices with the same driver should have identical or close signatures. This is an interesting characteristic to test the approach, since this is an unfavorable case to identify unique devices for the original fingerprinting method. To assess the improvements brought by the diversity approach, we build special datasets and evaluate the results of the fingerprinting process with the same metrics.

2.4.1 Datasets

One important remark is that the lack of data is a major issue in existing fingerprinting studies. We cannot make meaningful conclusions of the evaluation of a statistical method fingerprinting drivers, based on e.g. five different drivers. Moreover, comparing two different methods giving close results with two different datasets composed of few drivers is not fair. We try to overcome this by collecting as much data as possible and comparing methods at least with almost identical datasets (we use the same drivers even if capture methods differ, see Appendix A). The restriction we get is that to perform supervised learning, we have to know and to label the driver used by each device in the traffic capture (see Section 1.2.2). We thus have to control the devices and we can't just capture any traffic.

We build two sets of devices – *mixed* and *unique* – to evaluate different aspects of the methods. We make one traffic capture per device and per method tested, and we obtain four datasets: *mixed_franklin* and *unique_franklin* for Franklin et al. method, and *mixed_cache* and *unique_cache* for Cache's method. For Franklin et al. method, *mixed_franklin* is composed of 19 configurations with unique tuples {machine, NIC model, driver} and 9 different drivers; for Cache's method, *mixed_cache* is composed of 17 configurations because some devices could not be used and 9 different drivers. The *unique* set is composed of 9 Netgear USB adapters and a single driver for both methods. Table 2.1 summarizes their respective characteristics, and details can be found in Appendix A. These sets of devices give interesting perspectives in their composition: *mixed* set best characterizes Franklin et al. and Cache's method ability to identify drivers with its large set of drivers and configurations (same chipset and different drivers or same driver and different chipsets), while *unique* set best characterizes the methods' ability to identify unique devices in an unfavorable case: nine identical devices.

Given the lack of data, we make the choice to build a simulation for the two methods tested. The objectives are twofold: (1) testing the range of perturbations before implementing them, and (2) verifying the stability of the results by running several times the simulation.

2.4.2 Metrics

To be able to compare different methods, we use the same metrics for each experiment during the evaluation process.

True positive rate The usual metric in fingerprinting articles is the True Positive Rate (TPR): it is the ratio of well-classified devices over the total number of devices. It straightforwardly applies to fingerprinting methods that use the notion of distances, and match the device in the test set to the nearest device in the learning set. There generally is only one device at the nearest distance (thus one object per class), and the result of the classification is then binary: succeeds or fails.

Recall and precision In the case where there may be several devices from the learning set at the nearest distance of the device from the test set, we borrow metrics from the Information Retrieval domain. The *recall* is defined as the ratio of the number of relevant documents (here, devices) returned by a query to the total number of relevant documents in the collection. This measure answers the question “Are all the relevant documents retrieved?”. The *precision* is the ratio of the number of relevant documents returned by a query to the total number of documents returned by a query. This measure answers the question “Are the retrieved documents relevant?”. In a multi-classes environment, the global recall and precision are the mean recall and mean precision of all the classes. To compare different methods that have different recall and precision, another metric is used that combine both, the F-measure:

$$\text{F-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

In our context, we define the recall and precision, for one class being a unique device *device* (depicted by its MAC address). Let $correct_{device}$ be the number of samples correctly classified as belonging to *device*:

$$\text{recall}_{device} = \frac{correct_{device}}{\text{number of samples actually belonging to } device},$$

$$\text{precision}_{device} = \frac{correct_{device}}{\text{number of samples classified as belonging to } device}.$$

We also make an evaluation that characterizes the ability of the method to identify drivers. We can adapt the metrics above and calculate the recall and the precision for one class being a driver *driver*. Let $correct_{driver}$ be the number of samples correctly classified as belonging to *driver*:

$$\text{recall}_{driver} = \frac{correct_{driver}}{\text{number of samples actually belonging to } driver},$$

$$\text{precision}_{driver} = \frac{correct_{driver}}{\text{number of samples classified as belonging to } driver}.$$

If there is only one device at the nearest distance, then we have:

$$recall = precision = F\text{-measure} = TPR.$$

Chapter 3

Diversity in Cache’s method

The first method we examine is based on a statistical analysis of the *duration* field. We give an overview of the functioning of carrier sensing in 802.11, we detail the fingerprinting method by Cache, and we explain how we add diversity in the method. We provide experiment results: our best result so far is a TPR of 67% without diversity and 100% with.

3.1 Background: carrier sensing in 802.11

802.11 MAC uses Carrier Sense Multiple Access (CSMA) for the devices to transmit data in the shared transmission medium. CSMA exists in two flavors: (1) Carrier Sense Multiple Access with Collision Detection (CSMA/CD) where the devices stop their transmissions when detecting a collision and retry later; (2) Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) where the devices sense the channel and defer their transmissions for a random backoff time if it is busy. Because of the hidden nodes issue [15], 802.11 devices cannot detect collisions and therefore use CSMA/CA.

In almost every packet, a *duration* field is set to announce how long in microseconds the device intend to keep the channel. The field is a 16 bits value, which represents up to 65535 microseconds, but the standard doesn’t admit values over 32767. For fragmented data, the duration includes the time for the remaining fragments in addition to inter-frame spacings. For unfragmented data, it includes the inter-frame spacing and the time for an ACK. When no more traffic is needed – this is the case for management frame and some control frames such as ACK – the field is set to 0.

3.2 Cache’s method

Cache [6] uses the duration field to fingerprint 802.11 drivers. He observes that even though the field is 16 bits wide it generally takes only a few discrete values and that some implementations differ with distinctive values (sometimes illegal values). We first describe Cache’s original method, then our implementation.

3.2.1 Cache’s original method

Building the signatures

The signatures are composed of the duration values and the ratios of these durations in data and management frames. The method comprises two types of signatures:

1. tuples $(dur, ratio)$, where dur is the duration value and $ratio$ the ratio for this duration relative to the total number of packets;
2. tuples $(type, dur, ratio)$, where $type$ is the packet type, dur the duration value and $ratio$ the ratio of the tuple $(dur, ratio)$ relative to the total number of packets of type $type$.

These ratios may change depending on the device’s traffic for the same implementation, but Cache shows that it is stable enough to improve the algorithm. This statement is discussed in Section 3.5.

Learning stage

The learning algorithm compares each signature of the test set to each signature of the training set. It then produces a list of the learning set’s entries ordered by decreasing degree of match. It can use one of the two types of signatures or combine both. To compare two signatures, Cache introduces five different metrics. The first, *SimpleCompare*, compares the ratio of durations that only are in the intersection of the compared signatures. The second, *MediumCompare*, takes into account the uniqueness of some duration values with respect to all the signatures of the base: a discriminant duration would play a larger role. The third, *ComplexCompare*, only solicits the duration fields that are not part of the intersection of the compared signatures. The last two, *BayesCompare* and *ModifiedBayesCompare*, are based on Bayesian probabilities.

Evaluation

Cache uses traffic samples that are composed of 13 implementations (an implementation = a chipset + a driver) with 10 different drivers. The captures of each implementation’s traffic have been taken on four different WLANs three times.

Each metric is evaluated in the same way with Cache’s own method, which uses a ranking system. For each implementation I and matching metric M , and for the three samples s_1 , s_2 and s_3 , Cache defines the success probability $R_I = [(13 - s_1rank) + (13 - s_2rank) + (13 - s_3rank)] / (3 * 13)$, where $s_i rank$ is the rank assigned by M to the implementation I that actually produced the sample s_i . The success probability thus decreases with the rank attributed by M , and is not binary (succeeds or fails). The success rate of M relatively to all the implementations is the mean of all implementations’ success probability.

Cache’s results show that *MediumCompare* performed best using signatures including packet type ($R = 97\%$), followed closely by *SimpleCompare* (including packet type: $R = 97\%$, duration values only: $R = 94\%$). We choose to implement *SimpleCompare* using signatures with duration values only for its simplicity and good results.

3.2.2 Cache’s method revisited

We now discuss about the implementation that we make of Cache’s original method, regarding distance measures and evaluation. This implementation allows strict execution of Cache’s method, or a slightly improved version. The changes are not major, but rely on more standard tools and improve the results, in contrast to the tools used by Cache.

Standard histogram distance measures For the matching algorithm, Cache doesn’t use standard distance measures, but doesn’t clearly justify his choice. In particular, his first two similarity measures between two signatures restricts to the duration values in the intersection of both signatures. The role of discriminating values is then weakened. His third similarity measure restricts to the duration values that *are not* in the intersection of both signatures.

As a result, in his own evaluation, the results of this similarity measure are not as good as the first two. We implement Cache’s distance measure *SimpleCompare*. We also implement distance measures from the literature like Euclidean distance, Manhattan or Jaccard [7]. All these distance measures consider all the duration values in the union of both signatures.

An other distance measure We elaborate a distance measure – that we call *Flat* – that wouldn’t take into account the ratios, but the duration values only. The *Flat* distance measure yields good performances when the ratios are not stable; a full justification is given in Section 3.5. Two signatures A and B can then be considered as sets of values. $|A \cap B|$ is the cardinal number of the intersection of A and B , and $|A \cup B|$ the cardinal number of the union of A and B . We define d_{Flat} as:

$$d_{Flat} = 1 - \frac{|A \cap B|}{|A \cup B|}.$$

Another interesting property of the *Flat* distance measure is that we would need less traffic to build the signatures: one duration value would only have to be seen one time by the fingerprinter to be part of the signature. We would not need any additional traffic to build a consistent histogram.

Evaluation process We finally change Cache’s evaluation process to be able to compare the results from one method to the other. We use the metrics defined in Section 2.4: recall, precision and F-measure. The recall and precision are useful in this case when more than one signature of the training set are at the minimal distance of a signature in the test set. Yet, when only one signature is at the minimal distance, we have recall = precision = F-measure = TPR. We evaluate the ability of the fingerprinting method to identify the devices’ drivers (the original purpose of the method), but also the unique devices, depicted by their MAC addresses.

3.3 Cache’s method without diversity

The results of the evaluation of Cache’s method on *mixed_cache* dataset are presented in Table 3.1, and on *unique_cache* dataset in Table 3.2. We observe that Cache’s distance measure *SimpleCompare* performs badly in comparison to the other distance measures: 50% accuracy when evaluating driver identification, 29% accuracy when evaluating unique devices identification. With his metrics, Cache was able to get a success rate of 94%. The difference between our results and his is due to the difference in the evaluation method: our method is more binary – the classification succeeds or fails – whereas Cache’s evaluation method involves a ranking system. Moreover, we don’t make the evaluation with the same dataset than he does.

Driver identification The results of the evaluation of *mixed_cache* dataset are similar to Cache’s own results (more than 90% in success rate) with the four distance measures that we add: the F-measure is 89% for driver identification with Euclidean, Jaccard and Manhattan distance measures, and 93% with the *Flat* distance measure. The *Flat* distance measure is to be remarked for its recall of 100% for driver identification. It means that the prediction of one particular driver often return several drivers, but that there is always the good one in the returned list. It comes at the price of a lower precision (87%), but the F-measure that combines recall and precision is still better for this distance measure (93%). The duration field is thus an efficient feature to fingerprint drivers.

distance measures	drivers			unique devices		
	recall	precision	F-measure	recall	precision	F-measure
SimpleCompare	50%			29%		
Euclidean	89%			71%		
Jaccard	89%			71%		
Manhattan	89%			82%		
Flat	100%	87%	93%	100%	66%	80%

Table 3.1: Evaluation of Cache’s method without diversity on *mixed_cache* dataset. For *SimpleCompare*, Euclidean, Jaccard and Manhattan distance measures, we only have one signature in the training set at the minimal distance for all the devices in the test set, thus recall = precision = F-measure = TPR.

distance measures	unique devices		
	recall	precision	F-measure
SimpleCompare	56%		
Euclidean	56%		
Jaccard	56%		
Manhattan	67%		
Flat	100%	22%	36%

Table 3.2: Evaluation of Cache’s method without diversity on *unique_cache* dataset. For *SimpleCompare*, Euclidean, Jaccard and Manhattan distance measures, we only have one signature in the training set at the minimal distance for all the devices in the test set, thus recall = precision = F-measure = TPR.

Device identification We also evaluate unique device identification on *mixed_cache* dataset. For Euclidean and Jaccard distance measures, the F-measure is 71%, and for Manhattan 82%. The *Flat* distance measure gives again 100% in recall, and a lower precision (66%) which gives a F-measure of 80%. It actually is an interesting characteristic for the experiment: the method classifies so well drivers that when classifying a device, the result is often other devices with the same driver. We then evaluate the method on a tougher dataset: *unique_cache* dataset, composed of the nine USB adapters. Only unique devices identification is evaluated since all the adapters are run with the same driver. Here, *SimpleCompare* performs as well as Euclidean and Jaccard distance measures, with a F-measure of 56%. The *Flat* distance measure still has 100% in recall, but has virtually no precision (22%): when asked to predict a device, the classifier answers nearly all the devices in the training base. Its F-measure is then low (36%). We thus have a room for improvement with the diversity approach.

3.4 Cache’s method with diversity

We test fingerprint diversity by (1) simulating it a posteriori in existing traces, and then by (2) actually changing the behavior of devices. We investigate two approaches: injecting new frames with specific durations in signatures, and changing the duration fields of existing frames. We simulated both with encouraging results. Only the first approach could be implemented, and it also reveals good results. We evaluate the identification of unique devices on the nine Netgear USB, since it is the most unfavorable case and they run a single driver. The evaluation of our experiments are summarized in Table 3.3 page 22.

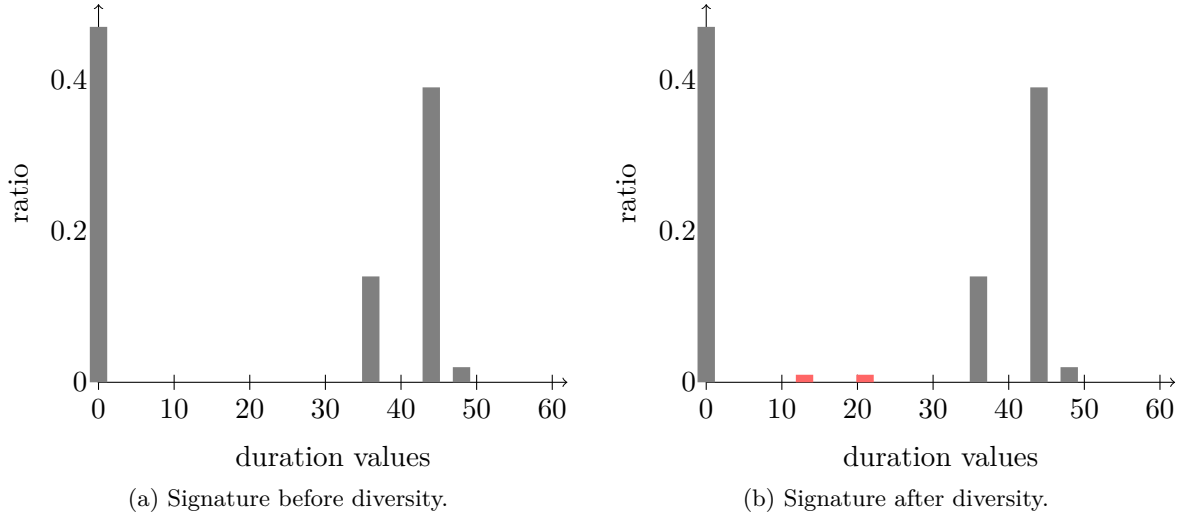


Figure 3.1: Injecting new durations: before and after diversity.

3.4.1 Approaches

Injecting new frames with specific durations The first approach is to inject some frames with specific duration values in the traffic from the fingerprintee. We then add some particular durations to the signature. This approach is illustrated in Figure 3.1. However, we must be careful to not add excessive durations that would impact the normal behavior of the network. Indeed, the durations indicate the channel reservation time, it shouldn't be much longer than it really needs. We also have to choose the type of the packet injected: if it requires an answer, it must be coherent with the device's state. We choose to inject probe requests, that can be legitimately sent by a device associated or not. As such, the packet type does not impact the signature, since we only take into account the duration.

Changing duration fields of existing frames The second approach we take is to fix for each device d_i a random number α_i in the interval $[1, n]$, and to systematically add it to all the duration fields of d_i 's outgoing frames. This approach is illustrated in Figure 3.2. However, in real life the limits of the interval would also have an impact on the normal behavior of the network.

3.4.2 Simulation

We make simulations of both of our diversity approaches by modifying *unique_cache* trace file a posteriori. We reasonably assume that one MAC address corresponds to one device.

Injecting new frames with specific durations We simply add to the existing trace file new frames with a specific duration and the source's MAC address, that would be interpreted by our implementation of Cache's method. In this simulation, we can configure the number of different durations and the number of repetitions of the same duration. The durations are randomly picked in the interval $]0, 44[\cup]44, 50[$. We choose 50 as an upper bound because it is in the order of magnitude of common frames' durations. We deliberately remove the durations 0 and 44 from the interval because it is common to nearly every driver. We run 10 times the simulation with 1, 2 and 10 different durations, and evaluate each simulation with the five distance measures. We inject each duration four times: we have to make sure that each unique

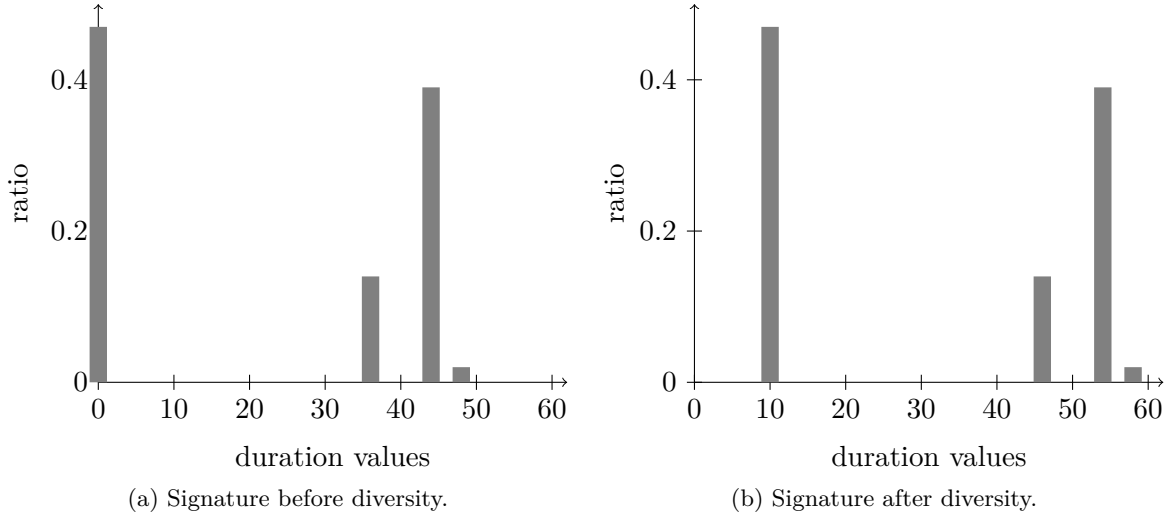


Figure 3.2: Changing duration fields: before and after diversity.

duration would be present in the learning set and in the test set. The results of the evaluation can be found in Table 3.3 for 2 different durations. The injection shows a degradation of the results for the *SimpleCompare* distance measure. Euclidean and Jaccard distance measures don't show any improvement even for 10 durations injected. This is due to the ratios of the new durations: they are very small compared to the others, and don't make a considerable difference. In contrast, Manhattan distance measure shows no improvement for one or two durations injected, but has 99% in TPR for 10 durations injected: this distance measure is more sensitive to little differences in the histograms. The distance measure that shows the greatest improvement is the *Flat* one: with the injection of 1 unique duration, we go from 66% without injection to 91% in precision (recall is still 100%), and with the injection of 2 different durations we obtain 100% in recall and precision.

Changing duration fields of existing frames We fix $\alpha_i \in [1, n]$ for each MAC address of a device d_i , and add α_i to each duration of d_i . The interval shouldn't be too large to avoid increasing too much the durations, and thus occupying the network for nothing. However, the amplitude of the interval also plays a role in the improvement of the results, and depends on the number of devices: with a large interval, we reduce the chances of collisions. There is thus a trade-off between the width of the interval and the improvement of the results. We run the simulations 10 times per interval and evaluate the simulation for each distance measure. The results of the evaluation can be found in Table 3.3 for an interval $[1, 10]$. The results increase when we can choose α_i in a bigger interval. *SimpleCompare*, Euclidean and Jaccard distance measures almost have the same performances; Manhattan distance measure performs slightly better; and *Flat* distance measure doesn't show as much improvements as the others. The minor improvement with the *Flat* distance measure is due to the fact that it doesn't take into account the ratios of the duration values while it is the only way for two signatures to differentiate when they have the same durations values – this happens when $\alpha_i = \alpha_j$ for two devices d_i and d_j . For Manhattan distance measure, we achieve 92% in TPR with each $\alpha_i \in [1, 10]$, and 99% with each $\alpha_i \in [1, 25]$.

	diversity	distance measure	recall	precision	F-measure
simulation	injection	SimpleCompare	42%		
		Jaccard	51%		
		Euclidean	51%		
		Manhattan	65%		
		Flat	100%		
	change	SimpleCompare	91%		
		Jaccard	91%		
		Euclidean	91%		
		Manhattan	92%		
		Flat	100%	70%	82%
actual devices	injection	SimpleCompare	89%		
		Jaccard	89%		
		Euclidean	89%		
		Manhattan	100%		
		Flat	100%		
	change (not implemented)	-	-	-	-

Table 3.3: Evaluation of Cache’s method on the *unique* set of devices with our two approaches of diversity (injecting new durations and changing all durations fields), in simulation and with actual devices. For the second simulation evaluated with the *Flat* distance, we sometimes have more than one signature in the training set at the minimal distance of devices in the test set, thus we need recall and precision. For the rest, we have recall = precision = F-measure = TPR.

3.4.3 Changing the behavior of a device

We then investigate the way to reproduce these approaches by changing the behavior of actual devices. The experiments are made with the Netgear USB adapters that run with the open source `ath9k_htc` driver: we use the devices that are part of the *unique* set to be able to compare the results with the simulation.

Injecting new frames with specific durations The first approach involves injecting frames. For this matter, the Atheros AR9271 chipset in the Netgear USB adapters is compatible with the injection mode with `mac80211` drivers, including `ath9k_htc`. We use the libraries `lorcon2` and `Net::Lorcon2` for Perl to inject raw 802.11 packets. As in the simulation, we choose to inject probe requests frames with durations in the interval $[1, 44 \cup 44, 50]$. We choose to inject 2 different durations, and each frame with one specific duration is sent four times, to be sure that the duration is present both in the training and in the test set even if a frame is missed. The code can be found in Appendix D. The results can be found in Table 3.3. Contrary to the simulation, we see a large improvement with all distance measures: we have a TPR of 89% for *SimpleCompare*, Jaccard and Euclidean distance measures, and 100% for Manhattan and *Flat* distance measures. It should be noted that these results are not statistic, but quantitative. For that matter, we are limited by the dataset, but the experiment confirms the results of the simulation, and particularly the good results with the *Flat* distance measure. However, this approach doesn’t seem to be possible for every NIC: first, the card and the driver must be compatible with injection mode. Moreover, some cards seem to modify the raw packets that are about to be injected. For instance, the Atheros card AR9285 modifies the duration applied to probe requests. This approach thus gives good results but is not possible in all situations.

Changing duration fields of existing frames The second approach requires to modify the function that calculate the durations in `mac80211` drivers (see Appendix B). We find a function that seems to calculate the durations in `mac80211: ieee80211_frame_duration`¹. However, when doing several experiments with the drivers `ath9k` and `ath9k_htc` (which is the `ath9k` driver for USB devices), we find that our modifications don't change the duration fields, and that this function isn't called at all. Actually, the calculation of the duration field appears to be made in hardware when the rate control algorithm itself is made in hardware – which seems to be the case according to Corbett et al. [9]. It clearly is the case for `ath9k_htc`. For `ath9k`, another function seems suitable: `ath_pkt_duration`². This function is actually called when using the driver, but our modifications are not taken into account. The massive retro-engineering efforts of finding the right function and the uncertainty of the modifications' feasibility has made us stop looking in that direction.

3.5 Future work

In future work, we want to examine the implication of diversity from the attacker point of view. We also want to examine the impact of various features to the robustness of signatures.

3.5.1 Security perspective

We investigated a way to add diversity in Cache's fingerprinting method from the defensive point of view. As part as future work, we propose to investigate the attacks and the implication of diversity. However, we have some clues about how an attacker may try to forge the signature of a genuine device. A weak attacker that owns the same NIC and driver as its victim will be defeated by the diversity: the signature of the victim will be slightly different from its own. A casual attacker that attempts to modify his own driver might have the same problems that we encountered: durations calculated by the hardware and not the driver need massive retro-engineering efforts. Still, the attacker may inject frames above his own traffic just like we did. His signature would then be closer to the victim's signature, but not the same. A more powerful attacker may record traffic from its victim – that includes diversity – and replay it. If the employed distance measure includes the ratio of each duration value, the attacker's traffic will be more difficult to insert without changing the signature; if the employed distance measure doesn't include the ratios, the attacker only needs to assure that his traffic doesn't contains values other than those of the legitimate signature.

3.5.2 Robustness of the signatures

Signatures of legitimate devices should be stable over time, or at least, the changes have to be taken into account in the signature base. During our study, we isolated three main parameters that seem to influence Cache's signatures: the Basic Service Set (BSS) parameters, the fingerprintee's traffic and its environment and hardware.

BSS parameters Cache points out that the BSS parameters influence the implementations' signatures, since they intervene in the calculation of the duration field. First, 802.11b adds a *short preamble* using a 56 bits field *sync* instead of the 128 bits one [15]: the length of the packet changes, and so is its duration. Moreover, 802.11g adds a *short slot time* to reduce the time before retransmitting a packet [15]. Finally, for two packets of the same length, the *data rates*

¹file `net/mac80211/util.c`

²file `drivers/net/wireless/ath/ath9k/xmit.c`

duration value	ratio
0	22%
36	14%
44	37%
48	2%
202	0%
314	25%

(a) Signature in scenario 1

duration value	ratio
0	14%
36	10%
44	64%
48	1%
202	0%
314	11%

(b) Signature in scenario 2

Figure 3.3: Influence of the amount of traffic on signatures, with an Atheros AR5B95 chipset on Windows 7.

at which it is sent change the duration. A signature is therefore bound to a set of parameters, but not to a particular access point.

Nature and amount of traffic The nature and amount of traffic also influence the signatures. Depending on the nature of the traffic, a station sending little frames won't necessarily have the same duration values in its signature as the same station sending bigger frames. Moreover, a station sending a very limited amount of data will tend to send more broadcast packets – for which durations are theoretically 0 – than data packets. It thus changes the ratios of the different durations. Cache found out that the ratios are stable enough to improve the fingerprinting algorithm. However, we observe that the differences could be quite important between two signatures of the same implementation. For instance, we capture traffic from the same station (Dexter, see Appendix A) in two different ways. In the first scenario, the station was not associated for 10 minutes, then it associated and waited for 2 minutes, sent 2 `wget` requests on `www.google.fr`, and stayed idle for a remaining 3 minutes. The second scenario was the same, but instead of 2 `wget` requests, the station sent 100. Both captures lasted 15 minutes. The signatures can be found in Table 3.3. The ratios for several duration values changed: for 0, 36, and 314 it decreased, and for 44 it increased. We can observe that if the ratio changed, the discrete values stay the same. The *Flat* distance measure introduced in Section 3.2.2 exploits this property. In the evaluation of the method with *mixed_cache* dataset, we remark that the *Flat* distance measure performs better than the other distance measures for the identification of drivers, the results are only slightly lower than with Manhattan distance measure for the identification of unique devices, and still better than *SimpleCompare*, Euclidean and Jaccard distance measures. The amount of traffic therefore has an impact on signatures.

Environment and hardware The environment also has an influence on the signatures: stations change their speed in response to the link quality. In a busy environment, there are more frames that require retransmission and the chip reduces the data rate to compensate, whereas in a quiet environment the chip sends frames faster. The increase of the data rate means a decrease of the durations for frames with the same length. However, we haven't been able to quantify the effects on the matching process, since it requires a perfect control of the environment. The duration field is therefore very tight to the hardware and it needs further investigations if the method is used to fingerprint drivers.

	without diversity			with diversity		
	recall	precision	F-measure	recall	precision	F-measure
SimpleCompare	56%			89%		
Euclidean	56%			89%		
Jaccard	56%			89%		
Manhattan	67%			100%		
Flat	100%	22%	36%	100%		

Table 3.4: Comparison of Cache’s method on the *unique* set of devices, without and with diversity (injection approach), evaluated with actual devices.

3.6 Conclusion

We presented Cache’s method that fingerprints 802.11 drivers using the duration field. We introduced more standard tools to Cache’s method and improved the results on a varied dataset. We also detailed two approaches to add diversity in this method, including one that have been successfully implemented on actual devices: the injection of new frames with specific duration values. We evaluated it on a dataset composed of nine identical devices and we achieved a TPR of 100% for two different distance measures. These results are summarized in Table 3.4. We therefore showed the interest of diversity coupled to Cache’s method for the identification of unique devices.

Chapter 4

Diversity in Franklin et al. method

The second method we examine is based on the procedure of active scanning. We first give an overview of this procedure in the standard, then we describe the fingerprinting method by Franklin, and we finally explain how we add diversity in the method and our results.

4.1 Background: active scanning in 802.11

To receive network services, a station must authenticate and associate with an access point. There are two methods, passive and active scanning. Here we only focus on active scanning that has been designed to be faster than passive scanning and that is the default mode on most implementations. Active scanning employs probe request frames (that are management frames) to scan an area for a wireless access point compatible with the data rates the station can support. If the access point is compatible, it sends a probe response frame and the station can associate with it. The general shape of the active scanning algorithm is known, but the details and the time constraints are left to the implementation and are generally undocumented.

In the IEEE standard [1], the procedure of active scanning is defined in the 802.11 MAC layer. When selecting a new channel, the algorithm first waits for a certain period and determines if it may transmit. It sends a probe request, starts a timer and senses if the channel is busy. If it is idle, it remains until the timer reaches a minimum time; else it remains until the timer reaches a maximum time and processes the probe responses. It finally scans the next channel. More precisely:

1. Select a channel
2. Wait an indication of an incoming frame or until the ProbeDelay time has expired;
3. Perform the Basic Access procedure to determine if the station may transmit;
4. Send a probe with the broadcast destination, the SSID, and the broadcast BSSID;
5. Clear and start a ProbeTimer;
6. If PHYCCA.indication (busy) is not detected before the ProbeTimer reaches MinChannelTime, then clear NAV and scan the next channel, else when ProbeTimer reaches MaxChannelTime, process all received probe responses;
7. Clear NAV and scan the next channel;

with ProbeDelay the delay prior to a transmission of a probe request on a new channel, MinChannelTime (resp. MaxChannelTime) the minimum time (resp. maximum time) spent on a channel.

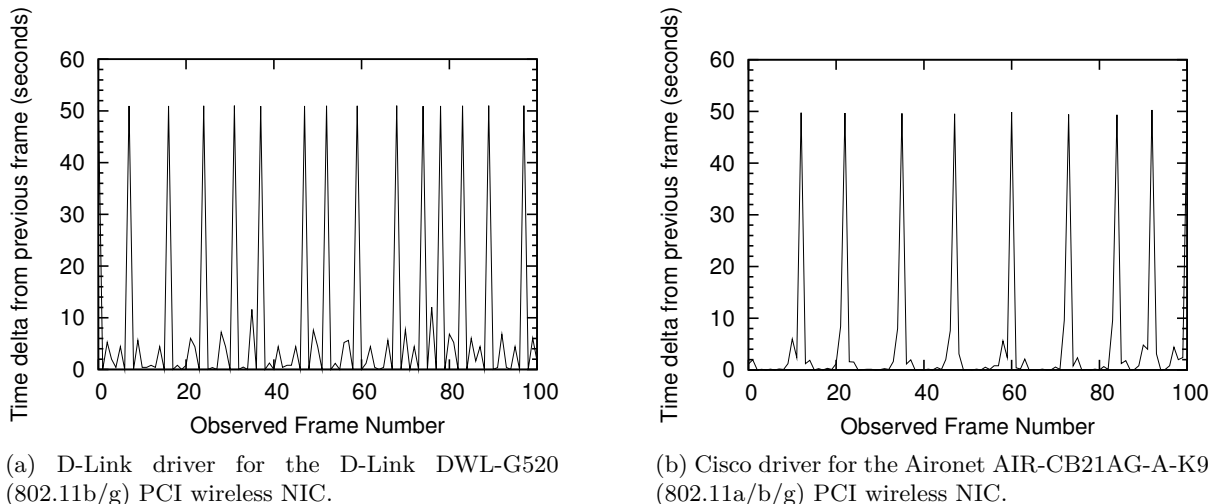


Figure 4.1: Plots of inter-frame timing of probe requests for two different drivers, from [14].

4.2 Franklin et al. method

Franklin et al. employ active scanning for fingerprinting in [14]. In the remaining of the report, we will refer to this method as “Franklin et al. method”. Corbett et al. also use active scanning, along with spectral analysis in [8]. Desmond et al. refine the work of Franklin et al. in [11]. Franklin et al. method is based solely on active scanning timings and is supposed to fingerprint the driver.

Building the signatures To build the signatures, Franklin et al. calculate the time between two consecutive probe requests. The signature is next generated by binning: the interval of continuous data points is translated into discrete bins of equal interval size to average out the noise. Two contributing attributes are isolated: the percentage of the inter-arrival time in each bin, and the average of the actual inter-arrival time values of the probe requests placed in the bin. To calculate the bin probabilities, the actual delta arrival time value is rounded to the closest discrete bin value, with bin width size of 0.8 seconds (found by empirical optimization). Graphics are also generated, with the time between two probe requests in vertical scale, and probe requests numbers (*i.e.* probe requests in order of occurrence) in horizontal scale. An example is given in Figure 4.1.

The learning stage The learning stage is done with the k -nearest neighbors algorithm (k -NN, here $k = 1$): each signature in the test set is matched with the closest signature in the training set. The distance measure used is ad-hoc and considers the percentage and the average of the inter-arrival time in the bin.

The evaluation The method is evaluated with three test sets that include 17 different drivers, in three different environments. For each driver there is up to four configurations, depending on the association to an access point or not, and on the driver management (Windows drivers can be managed by windows or by an standalone program). In total, there are 57 different configurations, thus 57 signatures. For the first test set, the fingerprinter is 15 feet from the fingerprintees with a limited amount of traffic. The experiment is then repeated in more difficult conditions: the fingerprinter further with a wall between him and the fingerprintees (test set 2),

and the fingerprinter closer with a lot of background traffic (test set 3). Franklin et al. use the accuracy (in fact, a true positive rate) to evaluate their method. As we can expect, the environments affect the accuracy, but the technique remains reliable in all the environments tested, from a 96% identification accuracy in test set 1 to 77% in test set 3.

4.3 Franklin et al. method without diversity

Franklin et al. claim to fingerprint the devices' drivers. We want to know if two or more devices with identical drivers lead to indistinguishable signatures, and if the diversity approach can improve the results of this method.

Driver identification We evaluate Franklin et al. method on *mixed_franklin* dataset – see Section 2.4 for the details of the evaluation. The results are similar to his own evaluation, since we have a True Positive Rate (TPR) of 95% when evaluating the ability of the method to identify the driver, with 9 different drivers. The method actually classifies correctly the driver of 18 devices over 19. The remaining driver is an `ath9k` driver, classified as an `ath9k_htc` driver, when they actually share the same implementation of MAC `mac80211`. The active scanning procedure is thus an efficient feature to classify drivers and their MAC layer.

Device identification We also observe that with the 19 different unique devices and 9 different drivers on *mixed_franklin* dataset, the method still get to a TPR of 89% of unique devices' identification. We then evaluate the method on *unique_franklin* dataset, that are nine identical devices. In 4 cases out of 9 (TPR of 44%), the closest signature of a device in the test set is the signature of the same device in the training set, which is more than random guessing. The method seems to classify more than the driver – further experiments need to be done to determine the role of physical parameters – but we have a room for improvement on the identification of unique devices.

4.4 Franklin et al. method with diversity

We test fingerprint diversity by (1) simulating an active scanning process and adding diversity in it, and then by (2) actually changing the behavior of devices by modifying the driver. In both situations, we want to test if the diversity achieves a better recognition of unique devices on the nine Netgear USB. It is the most unfavorable case since they run with a single driver, `ath9k_htc`.

4.4.1 Approach

We first search for an attribute that could be changed in the driver code and that would influence the signatures of Franklin et al. method. As the active scanning procedure is implemented in the MAC layer, it is in `mac80211` for the `ath9k_htc` driver. We use the version 2.6.39-1 of the package `compat-wireless` (see Appendix B). One easily modifiable attribute is the time spent in the channel. It is a simple definition in the file `net/mac80211/scan.c`, line 27: `#define IEEE80211_CHANNEL_TIME (HZ/33)`, in Time Unit (TU)¹. The HZ variable is the frequency with which the system's timer hardware is programmed to interrupt the kernel². This implementation also suggests the impact of the operating system on active scanning. The time spent in the channel is then the attribute that we modify for diversity in Franklin et al. method.

¹The IEEE 802.11 standard describes the TU as “a measurement of time equal to 1024 μ s” [1].

²250 by default in Ubuntu 11.04 desktop edition; to verify: `cat /boot/config-<kernelversion> | grep HZ`.

4.4.2 Simulation

Probe requests' traces generation

We first planned to change the timings of probe requests in real traffic captures a posteriori but the modification of the time spent on a channel has too many implications: will there be more probe requests or will there be more time between two probe requests? We made a simulator to generate probe requests traces as we could observe ones monitoring a single channel: we hence have control over the attributes of the algorithm without changing the driver. Even if we don't perfectly understand the phenomena at this stage, we can still reproduce it empirically.

We base our simulation both on the standard and the observations of the traces we had, to define the main attributes that influence the active scanning. The code can be found in Appendix C. The attributes include the time spent on the channel, the time between two probe requests, the idle time between two channels, the number of SSIDs probed and the time between two bursts of probe requests for different SSIDs. Every attribute is tunable and the simulation achieves to represent the behavior of existing devices, based on the graphics of Franklin et al. method. We assume that the driver sends probe requests for a limited amount of time, and not a specific number of probe requests. We add a small random in the time spent on a channel and the time between two probe requests, to fit more to reality: there is not always the same number of probe requests in a burst, and the peaks are not always at the same height.

One limitation is that the simulation – and by extension our measurements – only considers one channel. We hence have to approximate that what happens on one particular channel happens on the other channels, particularly the time spent in these channels. If we consider the number of probe requests sent on a channel independent of the time spent by the device on this channel – which seems to be the case according to [17] – then the time spent by the device on a channel only influences the height of the peaks. The peaks – *i.e.* the time spent on the other channels – represent an average behavior of the other channels.

The contribution of diversity

To simulate diversity on the *unique* set of devices, we make four series of simulations, each run ten times to have meaningful results.

1. Firstly, we simulate the behavior of nine identical configurations: same time spent on one channel (0.33 s), same time between two probe requests (0.08 s), same idle time between two channels (8.87 s) – the only difference was the slight random added to have more realistic traces. We obtain 6% of success: the identical devices tend to have indistinguishable signatures.
2. Secondly, we simulate the behavior of nine slightly different configurations: same time between two probe requests (0.08 s), same idle time between two channels (8.87 s), but the time spent on the channel changed a little for each trace: we try several level of perturbations of the time spent in the channel to see the influence on the fingerprinting process:
 - (a) from 0.320 s to 0.336 s with a step of 0.002 s: we obtain a TPR of 41%;
 - (b) from 0.310 s to 0.360 s with a step of 0.005 s: we obtain a TPR of 68%;
 - (c) from 0.30 s to 0.38 s with a step of 0.01 s: we obtain a TPR of 93%.

We observe that the simulated identical devices are badly recognized without diversity. With diversity, the TPR increases with the perturbation of the time spent on the channel, up to 93%.

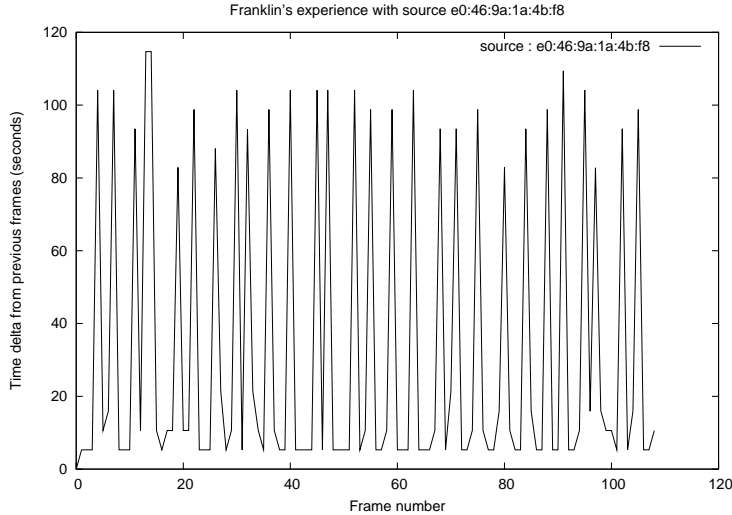


Figure 4.2: Netgear USB adapter, `ath9k_htc` driver modified with a `channel_time` of 1000 TU.

4.4.3 Changing the behavior of a device by modifying the driver

The experiment is made with one of the Acer netbook on Ubuntu (Mom, see Appendix A), with the package `compat-wireless`; the modifications in `mac80211` are then common to all `mac80211` drivers (see Appendix B). We first modify the variable responsible for the time spent in a channel (that we will later refer to as `channel_time`) to see the values that can be accepted, the results on the traffic captures and well-functioning. The original value is set to 7 TU, and we modify it to extreme values of 0 and 100 TU. We can directly observe this change of timing by looking at the USB adapter’s diode that lights up when it sends frames. At the signatures level, when `channel_time` increases, the time between two probe requests in a burst also increases and the time to return to the channel decreases.

For the experiment, we initialize `channel_time` with a random number between 0 and 100 TU, with a step of 2 TU between two random numbers. We then have a different value of `channel_time` for each of the nine Netgear USB adapter. A modification with such order of magnitude corresponds to the simulation 2(a), which is not the best case we test in simulation. Indeed, with this interval of values [0, 100] TU, there is too much risk of collisions with steps greater than 2 TU; on the contrary, if we take higher values for the interval, there is too much disruption in the normal behavior of the device. To assess the disruption, we modified `channel_time` to 1000 TU and observed the behavior (see Figure 4.2). The time spent in a channel by a device increased from less than a second for the original value, to more than a minute. It makes the active scanning procedure totally inefficient. Moreover, the signatures are very irregular with peaks from 80 to 110 seconds: the fingerprinting process could not be improved this way.

We then make traffic captures in the same conditions than the previous ones without diversity: one hour, non-associated. The results of the fingerprinting process show that 5 devices over 9 were well matched (TPR of 56%). Without diversity, it was 4 devices over 9 (TPR of 44%). With a dataset composed of only 9 devices, this increase is not statistically reliable and cannot be taken as a major improvement.

The lack of significant results can then be explained by two factors. Firstly, the perturbation of the attribute is limited, and so is its actual influence in the signatures: it does not change by far the height of the peaks, and it seems that the driver just spends more time to wait for responses – which is something we cannot see in traffic captures, and therefore in the signatures.

Secondly, with the increase of *channel_time* the values of the inter-arrival time of probe requests are less concentrated around a mean value. We then have more bins (the bins are 0.8 seconds) and the signatures or the distance measure might not be appropriate for these changes.

4.5 Future work

In future work, we want to examine the security of the fingerprinting method and the implication of the lack of diversity from the attacker point of view. We also want to examine the impact of various features to the robustness of signatures.

4.5.1 Security perspective

Given the lack of diversity for Franklin et al. method, an attacker that want to forge the signature of a genuine device falls into one of the two categories:

1. The attacker has the same driver than its victim, or the same MAC layer (see the results of driver identification in Section 4.3). Diversity cannot differentiate the victim's signature, thus the attacker's signature is the same. Attackers win.
2. The attacker has an other driver than its victim. A weak attacker under the same hypotheses than us might not be able to modify the driver more than we did. He would then have a different signature from its victim. A stronger attacker may be on passive scanning mode – which does not send any probe requests –, and may record traffic from its victim and replay it above its own traffic. He would then have the same signature than its victim. Weak attacker loses, strong attacker wins.

4.5.2 Robustness of the signatures

During our study, we observed three main parameters that seem to influence Franklin's signatures: the station's association or not, the SSIDs probed and the channel monitored.

Influence of association The active scanning procedure is designed for stations to find access points: if the station is not associated, it is used to discover and associate with an access point; if the station is associated, it is used in the handoff function, when a mobile station moves its association from one access point to an other. The behavior of associated devices actually differs from non-associated devices: some stop probing, others probe differently. Here we only focus on non-associated devices, the behavior of associated devices and its permanence need to be further explored. In a real life situation, the association has to be taken into account in the fingerprinting process.

Influence of SSIDs The observation of traffic captures highlights two main behaviors concerning SSIDs: some implementations only send undirected (broadcast) probes while other send directed probes. Whilst both are actually sent in broadcast, directed probes embed a specific destination SSID in their payload; only access points with this SSID respond with a probe response. Broadcast probes are ones with a null SSID; every AP with compatible data rates may respond. They lead to bursts of probe requests, then a change of channel. With directed probes, in addition to loop through the various channels, the algorithm loops through the various SSIDs for each channel probed. As some implementations wait up for a few seconds between directed probes for different SSIDs, it is directly reflected in signatures. While this is not part of the standard, this behavior can be explained by analyzing the `mac80211` code: the scan function for one channel is given in Appendix B.

Influence of channels The observation of the active scanning algorithm also highlighted some variations relatively to channels. Some implementations spend more time in a channel while sending as much probe requests as in an other channel: the time spent in a channel and the number of probe requests sent seem to be independent. Gupta et al. [17] point out that these parameters also vary with the channel, while Mishra et al. [26] highlight the correlation between the time between two probe requests and the number of probe responses. An other parameter is the visits received by the channel: before probing a channel, the standard asserts that devices should sense if it is empty or not. If not, it doesn't send any probe requests. We then have to consider the channel to monitor in the learning phase and in the prediction phase with this fingerprinting method.

4.6 Conclusion

We presented Franklin et al. method that fingerprints 802.11 drivers using the active scanning procedure. We detailed our approach to add diversity in this method, that consists in changing the time spent by the driver on channels. We implemented it and evaluated it on a dataset composed of nine identical devices. However, we didn't get any major improvement of the results: we couldn't modify the attribute enough without disrupting the normal behavior of devices, or the fingerprinting process itself.

Chapter 5

Future work

We have succeeded in the improvement of unique devices identification with Cache's method that originally fingerprints drivers. For Franklin et al. method, the difficulty resides in that the time spent in a channel could not be disrupted as much as we wanted in the driver code. While in simulation we successfully improved the results with an adequate perturbation, it degraded the behavior of the active scanning process with actual devices. What needs to be investigated is the role of the driver in this degradation, and if necessary the modification of an other attribute.

As part of a future work on diversity, we need to evaluate the ability of the methods that originally identify drivers to identify them anyway in these datasets. It means that for two different devices running the same driver, their signatures with diversity have to be distant enough to be considered as different devices in a dataset composed of other devices running the same driver; but it has to be close enough to be considered as the same driver in a dataset composed of other devices running other drivers. We move from a simple to a hierarchical classification problem. To be able to conclude on this question, we have to build an other dataset composed of several sets of several identical devices running the same drivers. Unfortunately, we were only able to build a single dataset of unique devices running the same driver.

An other complication of the diversity approach is its genericity. Indeed, we have to deal with a great variety of network cards and drivers. For Cache's method, we injected new frames: it requires to turn the network card in injection mode, which is not supported by every card. We also note that some cards modify the raw packets content before sending them: typically, the Atheros AR9285 card changed the durations of the raw packets, which lowered one approach on Cache's method. For Franklin et al. method, we modified a driver: most of them are proprietary and thus non modifiable.

Finally, we planned to test the diversity approach on Neumann et al. [27] method. This is still feasible, as we build the datasets for Cache's method from this perspective. The interest of this method is that it already is supposed to fingerprint unique devices, unlike the two methods that we explored.

Conclusion

In this report, we have shown the interest of 802.11 fingerprinting as a non-cryptographic security technique, and the importance of the improvement of fingerprinting methods. We implemented and evaluated a new approach called *diversity*, that improves unique devices identification, while keeping the existing methods as black boxes. In this approach, we slightly modify the attributes of the fingerprintees in a random way to differentiate them. We work in the scope of home network security – the defensive part – and we suppose that we have a certain control over devices.

We tested this approach on two state-of-the-art methods, that are Cache [6] and Franklin et al. [14] methods. The two methods are designed to fingerprint drivers. However, we observed that they perform better than random guessing in the identification of unique devices, with a room for improvement though. We first simulated diversity to explore the range of perturbation for one attribute and its implication on the classification process. Both methods show improvements. We then evaluated both methods on a dataset that is composed of nine actual and identical devices and we achieve 100% of success with Cache’s method. Yet, we obtain no major improvement for Franklin et al. method, due to the low range of perturbation allowed without disrupting too much the active scanning process.

In conclusion, diversity has proved to be able to improve the identification of unique devices on certain circumstances. We have to choose an attribute that is modifiable and can be largely perturbed without disrupting the usual device’s behavior.

Appendix A

Dataset

Devices

The experiments are made in the Technicolor Security and Content Protection laboratory – however, some traffic captures are made in a domestic environment. In addition to the uncontrolled devices that are part of the work environment, we have some special sets of controlled devices that comprise:

1. four Samsung NP-NC110 netbooks on Microsoft Windows 7, with the same configuration and the native Intel Centrino Wireless N-130 driver – which are named Richmond, Jen, Roy and Moss;
2. two Acer Aspire-One netbooks with the Atheros AR9285 chipset on Ubuntu 11.04, and the open source `ath9k` driver – which are named Mom and Hana;
3. one HP Compaq Mini netbook with the Broadcom BCM4312 chipset, on Fedora 16, proprietary Broadcom `wl` driver; and on Microsoft Windows 7, native Broadcom BCMWL6 driver;
4. two Belkin 54g wireless USB adapters with Belkin’s native driver – which are named Mireille and Josette ;
5. nine Netgear WNA1100 USB adapters, with the Atheros AR9271 chipset, on Ubuntu 11.04 on one of the Acer netbook and the open source `ath9k_htc` driver – which are named after the solar system’s planets (including Pluto).

We build two sets of devices to evaluate different aspects of the methods: *mixed* and *unique*. We make one traffic capture per device and per method tested, and we obtain four datasets: *mixed_franklin* and *unique_franklin* for Franklin et al. method, and *mixed_cache* and *unique_cache* for Cache’s method. For Franklin et al. method, *mixed_franklin* is composed of 19 configurations with unique tuples {machine, NIC model, driver} presented in Table A.1 and 9 different drivers; for Cache’s method, *mixed_cache* is composed of 17 configurations, because some devices could not have been used, and 9 different drivers. The *unique* set is composed of 9 Netgear USB adapters and a single driver for both methods. These two sets of devices give interesting perspectives in their composition: we have several strictly identical configurations (same chipset, same driver) to see the implication of the hardware in the fingerprinting process; we have configurations with the same chipset and two different drivers on different operating systems to see the implication of the chipset; and we have configurations with the same driver and different chipsets to see the implication of the driver.

Id	Label	OS / Network adapter / Driver
1	intel_dc:a9:71:54:cc:0c	Windows 7 Starter Intel Centrino Wireless N-130 driver Intel NETwNs32.sys v 14.1.1.3
2	intel_dc:a9:71:48:1a:66	idem
3	intel_dc:a9:71:54:b8:c0	idem
4	intel_dc:a9:71:53:37:3e	idem
5	atheros_0c:ee:e6:86:66:ae	Windows 7 Starter Atheros AR5B95 driver atheros v 8.0.0.177
6	ath9k_78:e4:00:05:19:94	Ubuntu 11.04 Atheros AR9285 driver ath9k kernel 2.6.35-32
7	ath9k_0c:ee:e6:b2:fa:5b	idem
8	belkin_94:44:52:02:dd:01_moss	Windows 7 Starter Belkin 54g driver belkin v 3.1.1.0
9	belkin_94:44:52:02:de:4d_moss	idem
10	belkin_94:44:52:02:dd:01_rich	idem
11	belkin_94:44:52:02:de:4d_rich	idem
12	bcmwl6_00:1f:3a:4c:69:14	Windows 7 Entreprise Dell Wireless 1390 WLAN Mini-Card driver Broadcom BCMWL6 v 5.60.18.8
13	bcmwl6_90:4c:e5:16:51:ed	Windows 7 Broadcom BCM4312 driver Broadcom BCMWL6 v 5.60.48.18
14	wl_90:4c:e5:16:51:ed	Fedora 16 Broadcom BCM4312 driver wl kernel 3.2.10-3
15	rtl8192su_00:08:d3:81:59:28	Windows 7 Hercules Wireless N Mini USB Key (RTL8191SU) driver Realtek RTL8192su.sys v1086.12.310.2010
16	rtl8192su_00:14:d1:b9:87:c1	Windows 7 Trendnet N150 TEW-648UB (RTL8188SU) driver Realtek RTL8192su.sys v1086.38.1125.2010
17	rtl8192su_00:08:d3:90:80:14	Windows 7 Hercules HWGUm-54 driver Realtek RTL8192su.sys v1086.38.1125.2010
18	ath9khtc_e0:46:9a:a5:6e:8e	Ubuntu 11.04 Netgear N150 Wireless USB adapter WNA110 (AR9271) driver ath9k_htc kernel 2.6.38-13
19	htc_64:a7:69:45:9a:0c	Android 2.2.1 HTC driver unknown

Table A.1: Detailed dataset.

Conditions of traffic captures

For each experiment, we use the same devices with the same drivers to be able to compare the methods. Nevertheless, the two methods we deal with don't classify devices using the same parts of the 802.11 standard: we then made different traffic captures.

Franklin's method For Franklin et al. method (see Chapter 4), we capture the 802.11 frames sent by each device non-associated, during one hour. To evaluate the method on our datasets, we split our traces in training and test set for each unique device with the first two thirds (in terms of number of frames) for the training set, and the last third for the test set.

Cache's method For Cache's method (see Chapter 3), we capture the 802.11 frames sent by each device according to a specific scenario that last 15 minutes: the station is not associated for 10 minutes, then it associates with a specific access point – that stays the same throughout the experiment – and waits for 2 minutes, sends 2 `wget` requests (`wget --spider www.google.fr`), and stays idle for a remaining 3 minutes. Since we made our dataset with an idea of sequentiality (first the device is not associated, then we associate it, etc.), we don't just cut the dataset in two parts. The learning (respectively the test set) is made with 2 consecutive frames out of 3 (respectively with 1 frame), etc. until the end of the capture.

Appendix B

Driver architecture and code

Driver architecture

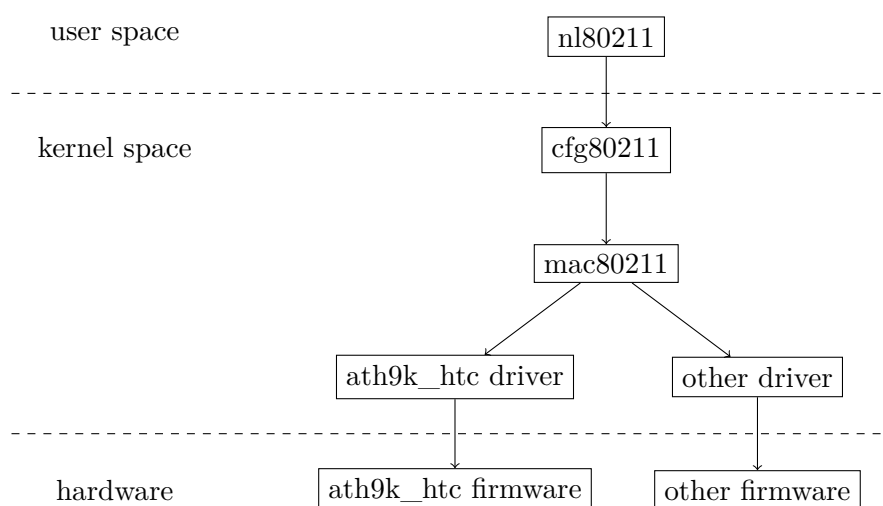


Figure B.1: ath9k_htc and mac80211 architecture.

For our experiments we used either native drivers of wireless devices (often provided with cards or USB adapters) or open source drivers. In this study we focus on the open source drivers, and specifically SoftMAC network devices's drivers, where the MAC Sublayer Management Entity (MLME) is implemented in software. The SoftMAC is loaded in Linux kernel as a module and implements the active scanning procedure. The two major implementations are `net80211` and `mac80211` [33]: `net80211` is the stack for MadWifi drivers (open source Atheros drivers); `mac80211` is now part of Linux kernel source tree and supports many drivers, including Atheros `ath9k`, `ath9k_htc`, `ath5k`, and Intel `iwlwifi`. The driver can be found in a package called `compat-wireless`, that gathers `mac80211` and `mac80211` drivers. Figure B.1 explains the architecture of `mac80211` and `ath9k_htc` which we work with.

Compiling and loading ath9k_htc

The procedure to compile and load the modified driver is the following, considering we're in the `compat-wireless` directory after decompressing the archive, with root rights enabled:


```

# modprobe -r ath9k_htc
# ./scripts/driver-select ath9k_htc
# make
# make install
# make unload
# modprobe ath9k_htc

```

Scan function

We give the code of the scan function for one channel in `mac80211` (file `net/mac80211/scan.c`), in `compat-wireless`, version 2.6.39-1.

```

650 static void ieee80211_scan_state_send_probe(struct ieee80211_local *local,
651                                             unsigned long *next_delay)
652 {
653     int i;
654     struct ieee80211_sub_if_data *sdata = local->scan_sdata;
655
656     for (i = 0; i < local->scan_req->n_ssids; i++)
657         ieee80211_send_probe_req(
658             sdata, NULL,
659             local->scan_req->ssids[i].ssid,
660             local->scan_req->ssids[i].ssid_len,
661             local->scan_req->ie, local->scan_req->ie_len);
662
663     /*
664     * After sending probe requests, wait for probe responses
665     * on the channel.
666     */
667     *next_delay = IEEE80211_CHANNEL_TIME;
668     local->next_scan_state = SCAN_DECISION;
669 }

```

Appendix C

Active scanning's simulation code

We give the code of our active scanning's simulation, introduced in Section 4.4.2.

```
1  #!/usr/bin/perl
2  use strict;
3  use warnings;
4  use POSIX;
5  use Getopt::Long;
6  # Clémentine Maurice
7
8  ## Usage
9  sub usage{
10 print STDERR << "EOF";
11 Simulation de probe requets\n
12 usage: perl $0 [-h -c nb_channel --ssid nb_ssid --inters inter_ssid] -f
    output_file -n name --max max_channel_time --inter inter_probe --idle
    inter_channel
13 --help (-h)           : this (help) message
14 --file (-f) output_file : output file containing the simulation
15 --name (-n) device_name : name of the device
16 --max max_channel_time : maximum time spent in a specific channel
17 --inter inter_probe    : time spent between two probe requests
18 --idle inter_channel   : idle time between two channels
19 --ssid nb_ssid         : number of SSIDs
20 --inters inter_ssid    : time between two bursts of probes for different SSIDs
21 EOF
22
23     exit ;
24 }
25 ## Variables declaration
26 my $usage = 0;
27 my $output = ""; # Output file
28 my $device_name = ""; # Device's name
29 my $nb_channel = 13; # Number of channels
30 my $min_channel_time = 0.15; # Minimum time spent in the chanel
31 my $max_channel_time = 0; # Maximum time spent in the channel
32 my $inter_probe = 0; # Time between two probe requests
33 my $inter_channel = 0; # Waiting time between two channels
34 my $nb_SSID = 1; # Number of SSIDs (by default, 1)
35 my $inter_SSID = 0; # Waiting time between bursts of probes for two SSIDs
36 my ($i, $j); # Iterator
37
38 # Retrieving arguments
39 GetOptions ('name|n=s' => \$device_name,
40 'file|f=s' => \$output,
41 'help|h' => \$usage,
```

```

42 'max=f' => \$max_channel_time,
43 'inter=f' => \$inter_probe,
44 'channel|c:i' => \$nb_channel,
45 'idle:f' => \$inter_channel,
46 'ssid:i' => \$nb_SSID,
47 'inters:f' => \$inter_SSID);
48 # Usage if asked or wrong arguments
49 if ($usage || $max_channel_time==0 || $inter_probe==0 || ($nb_SSID != 1 &&
50     $inter_SSID == 0)) { usage(); }
51
52 ## Opening output stream
53 open OUT, ">$_output"
54   or die "Echec de l'écriture dans le fichier $_output\n";
55 # First line
56 print OUT "\"No\\.\\.\", \"Time\\\", \"Source\\\", \"Destination\\\", \"Protocol\\\", \"Info\\\"\n";
57 my $time = 0; # Pseudo elapsed time
58 my $count = 0; # Number of probes
59 # Temps de la trace, en nombre de probes
60 while($count < 500) {
61     # Directed probes
62     for($j = 0; $j < $nb_SSID; $j++){
63         my $SSID_name = ($j == 0) ? "Broadcast" : "Reseau".$j;
64         # Positive or negative random
65         my $guess0 = int(rand(2)) % 2 ? 1 : 0;
66         # Adding some random
67         my $rand0 = $guess0 ? rand($max_channel_time/2) : -rand($max_channel_time/2);
68         # For one SSID and one channel
69         # Waiting in the channel $max_channel_time + random time
70         for($i = 0; $i < $max_channel_time+$rand0; $i=$i+$inter_probe) {
71             # Positive or negative random
72             my $guess1 = int(rand(2)) % 2 ? 1 : 0;
73             # Adding some random
74             my $rand1 = $guess1 ? rand($inter_probe/2) : -rand($inter_probe/2);
75             $count++;
76             # Probe requests at regular intervals
77             $time += $inter_probe+$rand1;
78             printf OUT "\"%u\\\", \"%f\\\", \"00:00:00:00:00:00_%s\\\", \"Broadcast\\\", \"IEEE_
79                 802.11\\\", \"Probe request, SSID=%s\\\"\n", $count, $time, $device_name,
80                 $SSID_name;
81         }
82     }
83     # If several SSIDs to proben waiting a certain time between
84     if ($nb_SSID > 1) {
85         $time += $inter_SSID;
86     }
87 }
88 # Simulation of visiting other channels...
89 # ... when there is only one SSID to probe
90 if ($nb_SSID == 1) {
91     $time += ($max_channel_time + $inter_channel) * $nb_channel;
92 }
93 # ... when there are several SSIDs to probe
94 # must add the time spent on each SSID + the time between two SSIDs
95 else {
96     $time += ($max_channel_time + $inter_channel + ($inter_SSID * $nb_SSID)) *
97         $nb_channel;
98 }
99 }
100 ## Closing output stream
101 close OUT;

```

Appendix D

Packet injection's code

We give the code of our packet injection, introduced in Section 3.4.3.

```
1  #!/usr/bin/perl
2  use strict;
3  use warnings;
4  use Net::Lorcon2 qw(:subs);
5
6  # Clémentine Maurice
7
8  ## Initialization
9  # Defining interface and driver
10 my $interface = 'ifconfig | grep -o 'wlan[0-9]*[mon]_';
11 $interface =~ s/[\n\s]+//;
12 my $driver = 'mac80211';
13 print "$interface\n";
14
15 # Setting lorcon2
16 my $lorcon = Net::Lorcon2->new(
17     interface => $interface,
18     driver => $driver,
19 );
20 $lorcon->setInjectMode;
21
22 ## Durations
23 # Choosing nb_durations the number of durations to add
24 my $nb_durations = $ARGV[0];
25 if(!defined $nb_durations) { die "usage: perl $0 nb_durations\n" ; }
26 my $j = 0;
27 # Choosing randomly different nb_durations in [1,50]
28 my @durations;
29 for($j = 0; $j < $nb_durations; $j++) {
30     do {
31         $durations[$j] = int(rand(49))+1;
32     } while ($durations[$j] == 44);
33 }
34
35 my @hex_durations;
36 my @string_durations;
37 my $i;
38
39 # Ugly stuff to pass the duration from big endian to little endian
40 # TODO Find something more elegant :)
41 for($i = 0; $i < scalar(@durations); $i++){
42     $hex_durations[$i] = sprintf("%x", $durations[$i]);
43     my ($byte1, $byte2);
```

```

44 # Duration in hexa is composed of five numbers or more : failure
45 if($hex_durations[$i] =~ /[a-fA-F0-9]{5,}/){
46     die "Wrong_duration_list\n";
47 }
48 # Duration in hexa is composed of four numbers
49 if($hex_durations[$i] =~ /([a-fA-F0-9]{2})([a-fA-F0-9]{2})/){
50     $byte1 = $1;
51     $byte2 = $2;
52 }
53 # Duration in hexa is composed of three numbers
54 elsif($hex_durations[$i] =~ /([a-fA-F0-9])([a-fA-F0-9]{2})/){
55     $byte1 = "0".$1;
56     $byte2 = $2;
57 }
58 # Duration in hexa is composed of two numbers
59 elsif($hex_durations[$i] =~ /([a-fA-F0-9]{2})/){
60     $byte1 = "00";
61     $byte2 = $1;
62 }
63 # Duration in hexa is composed of one number
64 elsif($hex_durations[$i] =~ /([a-fA-F0-9])/) {
65     $byte1 = "00";
66     $byte2 = "0".$1;
67 }
68 $hex_durations[$i] = $byte1.$byte2;
69 $string_durations[$i] = pack("h*", join("", map { scalar reverse $_ } unpack("(
    A8)*", $hex_durations[$i]]));
70 }
71
72 ## Packet definition
73 my $frame_ctl = "\x40\x00"; # Type of frame: probe request
74 my $DA = "\xff\xff\xff\xff\xff\xff"; # Destination address: broadcast
75 # Finding source address
76 my $SA_temp = `ifconfig -a $interface | grep -E -o '[:xdigit:]{2}(:[:xdigit
    :]{2}){5}'`;
77 $SA_temp =~ s/[\n\s]+//;
78 my @SA_temp = split(/:/, $SA_temp);
79 my $SA = pack("H*", join("", @SA_temp));
80 my $bssid = "\xff\xff\xff\xff\xff\xff"; # BSSID: broadcast
81 my $seq_ctl = "\xa0\x02"; # Sequence control: arbitrary number (12)
82 my $body = "\x00\x00\x01\x08\x02\x04\x0b\x16\x0c\x12\x18\x24\x32\x04\x30\x48\x60\
    x6c\x03\x01\x04\x2d\x1a\x6e\x11\x1b\xff\x00\x00\x00\x00\x00\x00\x00\x00\
    x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00";
83 ## Packet injection
84 # For each duration
85 my $k=0;
86 for($i = 0; $i < scalar(@string_durations); $i++) {
87     # Assembling packet
88     my $packet = $frame_ctl.$string_durations[$i].$DA.$SA.$bssid.$seq_ctl.$body;
89     # Sending packet four times
90     for($k = 0; $k < 4; $k++){
91         my $t = $lorcon->sendBytes($packet);
92         print "T:␣$t;␣duration:␣$durations[$i]\n";
93         sleep(1);
94         if (! $t) {
95             print "[-]␣Unable␣to␣send␣bytes\n";
96             exit 1;
97         }
98     }
99 }

```

Bibliography

- [1] IEEE Std 802.11-1997 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Technical report, IEEE Computer Society, 1997.
- [2] IEEE Std 802.11i-2004 - Amendment 6: Medium Access Control (MAC) Security Enhancements. Technical report, IEEE Computer Society, 2004.
- [3] C. Arackaparambil, S. Bratus, A. Shubina, and D. Kotz. On the reliability of wireless fingerprinting using clock skews. In *Proceedings of the 3rd ACM conference on Wireless network security (WiSec'10)*, pages 169–174, 2010.
- [4] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles. Active behavioral fingerprinting of wireless devices. In *Proceedings of the 1st ACM conference on Wireless network security (WiSec'08)*, pages 56–61, 2008.
- [5] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless device identification with radio-metric signatures. In *Proceedings of the 14th ACM international conference on Mobile computing and networking (MobiCom'08)*, pages 116–127. ACM, 2008.
- [6] J. Cache. Fingerprinting 802.11 implementations via statistical analysis of the duration field. *Uninformed.org*, 5, 2006.
- [7] S. H. Cha. Taxonomy of nominal type histogram distance measures. In *Proceedings of the American Conference on Applied mathematics (MATH'08)*, pages 325–330, 2008.
- [8] C. L. Corbett, R. A. Beyah, and J. A. Copeland. Using active scanning to identify wireless nics. In *Proceedings of the IEEE Information Assurance Workshop*, pages 239–246, 2006.
- [9] C. L. Corbett, R. A. Beyah, and J. A. Copeland. Passive classification of wireless nics during rate switching. *EURASIP Journal on Wireless Communications and Networking*, 2008:2, 2008.
- [10] B. Danev, D Zanetti, and S Capkun. On physical-layer identification of wireless devices. *ACM Computing Surveys*, 2011.
- [11] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee. Identifying unique devices through wireless fingerprinting. In *Proceedings of the 1st ACM conference on wireless network security (WiSec'08)*, pages 46–55, 2008.
- [12] J.P. “Johnny Cache” Ellch. Fingerprinting 802.11 devices. Master’s thesis, U.S. Naval Postgraduate School, September 2006.
- [13] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of rc4. In Springer, editor, *Selected areas in cryptography*, pages 1–24, 2001.

- [14] J. Franklin, D. McCoy, P. Tabriz, V. Neagoie, J. V. Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of the 15th USENIX Security Symposium*, pages 167–178, 2006.
- [15] M. S. Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition*. O’Reilly Media, Inc., 2005.
- [16] K. N. Gopinath, P. Bhagwat, and K. Gopinath. An empirical analysis of heterogeneity in ieee 802.11 mac protocol implementations and its implications. In *Proceedings of the 1st ACM international workshop on wireless network testbeds, experimental evaluation & characterization (WiNTECH’06)*, pages 80–87, 2006.
- [17] V. Gupta, R. Beyah, and C. Corbett. A characterization of wireless nic active scanning algorithms. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC’07)*, pages 2385–2390, 2007.
- [18] J. Hall, M. Barbeau, and E. Kranakis. Enhancing intrusion detection in wireless networks using radio frequency fingerprinting. In *Proceedings of the 3rd IASTED International Conference on Communications, Internet and Information Technology (CIIT’04)*, pages 201–206, 2004.
- [19] C. He and J. C. Mitchell. Security analysis and improvements for ieee 802.11i. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS’05)*, pages 90–110, 2005.
- [20] C. Idland. Detecting mac spoofing attacks in 802.11 networks through fingerprinting on the mac layer. Master’s thesis, Norwegian University of Science and Technology, June 2011.
- [21] S. Jana and S. K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. In *Proceedings of the 14th ACM international conference on Mobile computing and networking (MobiCom’08)*, pages 104–115, 2008.
- [22] J.M. Keller, M.R. Gray, and J. Givens Jr. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(4):581, 1985.
- [23] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.
- [24] G. Lackner and P. Teufl. Ieee 802.11 chipset fingerprinting by the measurement of timing characteristics. In *Proceedings of the Australasian Information Security Conference*, 2011.
- [25] G. Lackner, P. Teufl, and R. Weinberger. User tracking based on behavioral fingerprints. In *Proceedings of the 9th International Conference on Cryptology And Network Security (CANS’10)*, pages 76–95. Springer, 2010.
- [26] A. Mishra, M. Shin, and W. Arbaugh. An empirical analysis of the ieee 802.11 mac layer handoff process. *ACM SIGCOMM Computer Communication Review*, 33(2):93–102, 2003.
- [27] C. Neumann, O. Heen, and S. Onno. 802.11 device fingerprinting using frame inter-arrival histograms. Technical report, Technicolor, 2012.
- [28] C. Neumann, O. Heen, and S. Onno. An empirical study of passive 802.11 device fingerprinting. In *Proceedings of the 1st International Workshop on Network Forensics, Security and Privacy (NFSP’12)*, 2012.

- [29] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th ACM international conference on Mobile computing and networking (MobiCom'07)*, pages 99–110, 2007.
- [30] A. A. Samra and R. Abed. Enhancement of passive mac spoofing detection techniques. *International Journal of Advanced Computer Science and Applications*, 1(5), 2010.
- [31] B. Sieka. Active fingerprinting of 802.11 devices by timing analysis. In *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (CCNC'06)*, pages 15–19, 2006.
- [32] B. Sieka. Using radio device fingerprinting for the detection of impersonation and sybil attacks in wireless networks. *Proceedings of the 3rd European Workshop on Security and Privacy in Ad-Hoc and Sensor Networks*, pages 179–192, 2006.
- [33] M. Vipin and S. Srikanth. Analysis of open source drivers for ieee 802.11 wlans. In *Proceedings of the IEEE International Conference on Wireless Communication and Sensor Computing (ICWCSC'10)*, pages 1–5, 2010.
- [34] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
- [35] K. Zeng, K. Govindan, and P. Mohapatra. Non-cryptographic authentication and identification in wireless networks. *IEEE Wireless Communications Magazine*, 17(5):56–62, 2010.
- [36] X. Zheng, C. Chen, C. T. Huang, M. M. Matthews, and N. Santhapuri. A dual authentication protocol for ieee 802.11 wireless lans. In *Proceedings of the 2nd International Symposium on Wireless Communication Systems (ISWCS'05)*, pages 565–569, 2005.