Why MAC Address Randomization is not Enough:
An Analysis of Wi-Fi Network Discovery Mechanisms

Mathy Vanhoef[†], Célestin Matte[‡], Mathieu Cunche[‡], Leonardo S. Cardoso[‡], Frank Piessens[†]
[†]iMinds-Distrinet, KU Leuven , [‡]Univ Lyon, INSA Lyon, Inria, CITI, France

ABSTRACT

We present several novel techniques to track (unassociated) mobile devices by abusing features of the Wi-Fi standard. This shows that using random MAC addresses, on its own, does not guarantee privacy.

First, we show that information elements in probe requests can be used to fingerprint devices. We then combine these fingerprints with incremental sequence numbers, to create a tracking algorithm that does not rely on unique identifiers such as MAC addresses. Based on real-world datasets, we demonstrate that our algorithm can correctly track as much as 50% of devices for at least 20 minutes. We also show that commodity Wi-Fi devices use predictable scrambler seeds. These can be used to improve the performance of our tracking algorithm. Finally, we present two attacks that reveal the real MAC address of a device, even if MAC address randomization is used. In the first one, we create fake hotspots to induce clients to connect using their real MAC address. The second technique relies on the new 802.11u standard, commonly referred to as Hotspot 2.0, where we show that Linux and Windows send Access Network Query Protocol (ANQP) requests using their real MAC address.

1. INTRODUCTION

Tracking people through their mobile devices has become controversial but common. For example, leaked documents show the NSA tracks people's cell phone location, and later analyses this data under programs such as Co-Traveler to infer relationships between people [19]. Under the programs Gilgamesh and Shenanigans, captured cell phone locations are used to perform targeted drone attacks [41]. As a more commercial example, smart trash cans in the UK used Wi-Fi to track the movements of people, in order to gain insight into people's shopping behaviour [22]. This is possible because Wi-Fi-enabled devices routinely transmit probe requests to search for nearby networks, and these requests contain the unique MAC address of the device. An attacker can easily capture and track these requests. In response

to these privacy violations, most Operating Systems (OSs) have now implemented different variants of MAC address randomization. While a commendable initiative, we show that all implementations of MAC address randomization fail to provide adequate privacy.

First, we analyse the content of probe requests by focusing on Information Elements (IEs), which are used to communicate extended information on the device and its capabilities. Based on real-world datasets containing more than 8 million probe requests, we show that the number of elements, their value, and the order they are in form a fingerprint of a device (called the IE fingerprint). This IE fingerprint can be used to defeat MAC address randomization. In some cases, the IE fingerprint even uniquely identifies a device in the datasets. We also found that the Wi-Fi Protected Setup (WPS) element may leak the original MAC address of the device.

We continue by studying the sequence number field, which is incremented for each transmitted frame. We consolidate previous observations [18] that this field is not reset upon identifier change in current implementations of MAC address randomization. By combining the sequence number field with the IE fingerprint, we present an algorithm that tracks devices over time and thus defeats MAC address randomization. Based on simulations, we show that this algorithm can track a significant fraction of devices.

Inspired by the work of Bloessl et al. [6], we also analyze the scrambler seeds of commodity Wi-Fi devices. We find that this field in the 802.11 physical layer is predictable and can thus be used for tracking. As opposed to the sequence number field, the scrambler seed is managed by the hardware. Hence it is more difficult, if not impossible, to fix this unwanted predictability through software updates.

Finally, we introduce and analyze active attacks which reveal a target device's real MAC address despite randomization. This is done by creating fake Access Points (APs) that advertise either popular SSIDs, or the support of Hotspot 2.0. A station will reveal its real MAC address when connecting to, or respectively communicating with, our fake APs. By spoofing only 5 SSIDs, we were able to retrieve the MAC address of 17.4% of devices. The attack abusing the Hotspot 2.0 standard uncovered the MAC address of 5.2% of devices.

To summarize, our main contributions are:

- We study information elements in probe requests, and discover new fields and techniques to track users.

- We demonstrate that scrambler seeds of commodity Wi-Fi radios are predictable, and show that devices are trackable through this field.

- We show that advertising fake hotspots, in particular when combined with the Hotspot 2.0 protocol, can completely defeat MAC address randomization.

The remainder of this paper is organized as follows. Section 2 introduces relevant parts of the 802.11 standard, and datasets used throughout the paper. A privacy analysis of information elements in probe requests is done in Section 3, and in Section 4 we demonstrate how combining this with predictable sequence numbers can be used to track devices. In Section 5, we show that scrambler seeds of commodity devices are predictable. Section 6 introduces attacks based on fake APs and the Hotspot 2.0 protocol. Finally, Sections 7 and 8 discuss related work and conclude.

2. BACKGROUND

In this section, we introduce vendor implementations of MAC address randomization, relevant parts of the 802.11 physical layer, the Hotspot 2.0 standard, and used datasets.

2.1 MAC Address Randomization

To prevent third parties from using the MAC address to track devices, several vendors have implemented MAC address randomization. This follows the suggestion of Gruteser et al. [25] to use disposable interface identifiers in order to improve users' privacy. In practice, this implies that probe requests no longer use the real MAC address of the device. For example, a new MAC address can be used for each scan iteration, where one scan iteration consists of sending probe requests on all usable channels. However, since a (draft) specification on MAC address randomization does not yet exist, iOS, Windows, and Linux, all implemented their own variants of MAC address randomization. This raises the question whether their implementations actually guarantee privacy. In the remainder of the paper, we use randomization as a synonym of MAC address randomization.

2.1.1 iOS

Apple added MAC address randomization to its devices starting from iOS 8 [42]. In iOS 8, randomized addresses are only used while unassociated and in sleep mode [18]. iOS 9 was extended to also use randomization in what Apples calls location and auto-join scans [42]. Based on our own experiments, this means that randomization is now also used when the device is active, i.e., when the screen is turned on.

2.1.2 Android

Android 6.0 uses randomization for background scans if the driver and hardware support it [2]. Unfortunately, we did not have a device to test and verify this in practice.

Although Android versions before 6.0 do not support randomization, several applications supporting this feature have been released [9, 3]. Common features of those applications are a periodical update of the MAC address to a random value, but also the manual modification of this address by the user. Note that those applications require root privilege to operate, which reduce their impact for the average user.

2.1.3 Windows

Microsoft supports randomization since Windows 10 [45]. Enabling randomization is possible if the hardware and driver support it. Interestingly, not only does Windows use random addresses for probe requests, it also uses a random address
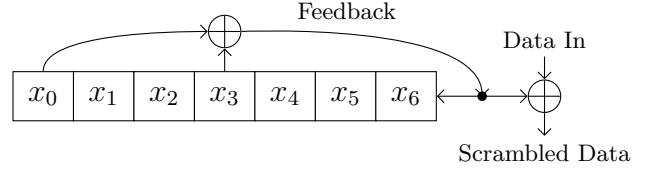

Figure 1: The scrambler used in 802.11 frames.

when connecting to a network. To assure the client always uses the same address when connecting to a particular network, a per-network address is calculated as follows [27, 28]:

$$addr = \text{SHA-256}(SSID, macaddr, connId, secret)[:6] \quad (1)$$

Here $SSID$ is the name of the network, $macaddr$ the original MAC address, and $connId$ a parameter that changes if the user removes (and re-adds) the network to its preferred network list. The $secret$ parameter is a 256-bits cryptographic random number generated during system initialization, unique per interface, and kept the same across reboots [28]. Bits in the most significant byte of $addr$ are set so it becomes a locally administered, unicast address. This hash construction is similar to the generation of IPv6 interface identifiers as proposed in RFC 7217 [21]. It assures that systems relying on fixed MAC addresses continue to work as expected, e.g., when authentication is performed based on the MAC address. Users can also manually instruct the OS to daily update the per-network address randomly.

2.1.4 Linux

Linux added support for MAC address randomization during network scans in kernel version 3.18. The address should be randomized for each scan iteration [24]. Drivers must be updated to support this feature. The `mvm` module of the `iwlwifi` driver supports randomization since kernel 3.18. The `brcmfmac` driver added support for this in kernel 4.5.

The privacy-oriented Linux distribution Tails [1] does not support MAC address randomization during network scans. Instead, it generates a (new) random MAC address at boot. This random address keeps the first 3 bytes of the original address, the Organization Unique Identifier (OUI), and only randomizes the last three bytes. While not as optimal as periodical address changes, it does prevent tracking over extended periods of time.

2.2 The Wi-Fi Physical Layer

The 802.11 standard defines two popular modulation techniques: Direct-Sequence Spread Spectrum (DSSS) and Orthogonal Frequency Division Multiplexing (OFDM). A disadvantage of OFDM is its high peak-to-average power ratio, increasing the bit-error ratio and out-of-band radiation [44]. This problem can be mitigated by using a scrambler that removes repetitive patterns in the data being modulated and transmitted. In 802.11, the scrambler XORs the input data with a bit sequence generated by a Linear Feedback Shift Register (LFSR) whose feedback function is [31, §18.3.5.5]:

DEFINITION 1. *The 802.11 scrambler* feedback *function* $L : \mathbb{F}_2^7 \to \mathbb{F}_2$ *is defined by* $L(x_0 x_1 \ldots x_6) = x_0 \oplus x_3$.

We call x_0 and x_3 the feedback taps. Here $\mathbb{F}_2$ is the field $\{0, 1\}$, and $\mathbb{F}_2^n$ a bitstring of length n representing LFSR states. Concatenation of bitstrings x and y is denoted by xy.
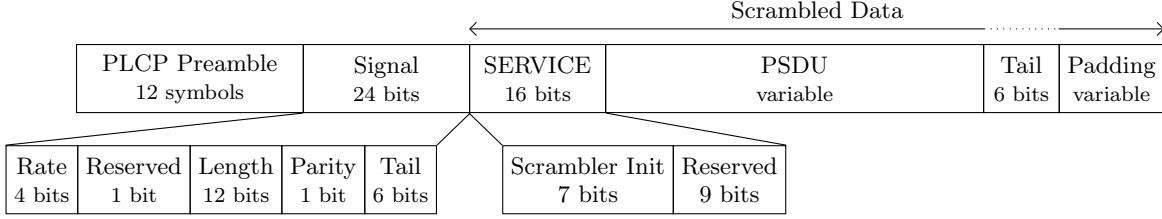
Figure 2: Format of legacy OFDM frames. The Tail field is zeroed and not scrambled. Bits are shown in transmit order.

For a bitstring $x \in \mathbb{F}_2^n$, x_i denotes the i-th bit of x (with $0 \leq i < n$). The shift function of the LFSR becomes:

DEFINITION 2. *The* shift *function* $S_L : \mathbb{F}_2^7 \to \mathbb{F}_2^7$ *is defined by* $S_L(x_0 x_1 \ldots x_6) = x_1 \ldots x_6 L(x_0 x_1 \ldots x_6)$.

The resulting LFSR is shown in Fig. 1.

The layout of (legacy OFDM-encoded) frames is shown in Fig. 2. Scrambling is done on all data bits starting from, and including, the SERVICE field (see Fig. 2). After scrambling, the tail field is overwritten with zeros. The scrambler is self-synchronizing. This means that the Scrambler Init field is initialized to all zeros, so the first 7 feedback bits will effectively be written to this field. Hence, the Scrambler Init field does not contain the scrambler seed, but the state of the LFSR *after* 7 shifts. Since probe requests are generally sent at the most reliable encoding available, DSSS is used in the 2.4 GHz band, and OFDM in the 5 GHz band. Surprisingly, DSSS frames use a fixed value for the scrambler seed [31, §17.2.4]. Only legacy 802.11 radios generate a random seed for DSSS-encoded frames. This makes the DSSS-encoded probe requests in the 2.4 GHz irrelevant in our attacks that rely on the scrambler seed. In contrast, OFDM-encoded frames use variable scrambler seeds. Therefore, when investigating the generation of scrambler seeds in Sect. 5, we will focus on probe requests transmitted in the 5 GHz band.

2.3 Hotspot 2.0

Hotspot 2.0 is an initiative of the Wi-Fi Alliance to streamline network discovery and selection, aiming to create a roaming experience matching that of cellular phones [46]. It allows clients to discover hotspots for which they have appropriate credentials, and provides automatic roaming between wireless networks. Hotspot 2.0 relies on 802.11u, a standard providing a communication channel even when the station is unassociated with an Access Point (AP) [32]. Stations use this channel to query an AP for network access information using the Access Network Query Protocol (ANQP). For example, ANQP can be used for discovering which credentials can be used to authenticate to a hotspot.

2.4 Datasets

Throughout our study, we used several datasets to pinpoint identifying elements contained in Wi-Fi frames and to evaluate the performances of our tracking attacks. The following datasets were used: the `Train-station` dataset captured around one large train station in Lyon in October 2015; the `Lab` dataset, a 5-day-long capture in October 2015 in our laboratory; and the `Sapienza` probe request dataset [4] that has been captured by Barbera et al. in 2013. Table 1 summarizes the characteristics of those datasets.

In order to limit privacy risks when analyzing the datasets,

Table 1: Details of the probe requests datasets.

Dataset	Lab	Train-station	Sapienza
#MAC addr.	500	10 000	160 000
#Probe Req.	120 000	110 000	8 million
Time frame	Oct '15	Oct/Nov '15	Feb/May '13
Location	Lab	Train Station	Rome

we restricted the capture to probe requests only, which means that no network data was collected. In addition, we applied to our datasets the same anomyzation method as used by Barbera et al. on the `Sapienza` dataset: once collected, all identifiers (MAC addresses and SSIDs) were replaced by a pseudonym, preventing any re-identification.

In all datasets we removed probe requests sent from locally administered addresses. These are either random MAC addresses, or specially assigned ones, and in general do not remain constant. Since we use MAC addresses as unique devices identifiers to check the performance of our algorithms, they would distort our results. Finally, based on sequence numbers and device-specific IEs, we detected and removed one device that kept the first three bytes of its MAC address, but randomized the last three.

3. PROBE REQUEST FINGERPRINTING

In this section, we study how much identifying information can be found in probe requests besides MAC addresses, timing, and sequence numbers. In particular, we study the data carried in the frame body of probe requests, and show that it can be used to fingerprint and identify devices.

3.1 Information Element Fingerprint

Probe requests include data in their frame body under the form of Information Elements (IEs) [31, §7.2.3], also called tagged parameters, or tags. These IEs are not mandatory and are used to advertise the support of various functionalities. They are generally composed of several subfields whose size can range from one bit to several bytes. We identify 12 useful elements, presented in Table 2. This list is not exhaustive and could be extended. Selected IEs include items related to `Supported Rates`, `High Throughput capabilities` and `Interworking Capabilities`. Because they are optional, those IEs are not included by all devices and the set of IEs can therefore vary from one device to another, depending on the configuration and capabilities of the device. While the 802.11 standard states that IEs must be sorted in ascending order based on their tag [31, §8.4.2.1], several devices ignore this and use a custom order. Therefore the order of IEs is also potential source of information.

Table 2: Analysis of the Information Elements of probe requests in the considered datasets. For each item: the entropy brought by the element, the percentage of devices for which this item is stable over time, and the percentage of devices that include this item in their probe requests.

Element	Entropy (bits)			Stability			Affected devices		
	Lab	Station	Sapienza	Lab	Station	Sapienza	Lab	Station	Sapienza
HT capabilities info	3.94	4.74	3.35	96.0%	95.9%	99.6%	90.9%	90.0%	81.1%
Ordered list of tags numbers	4.23	5.24	4.10	93.6%	94.2%	91.2%	100%	100%	100%
Extended capabilities	2.59	2.57	0.064	98.5%	99.4%	99.9%	55.4%	51.3%	0.6%
HT A-MPDU parameters	2.59	2.67	2.54	97.8%	99.1%	99.7%	90.9%	90.0%	81.1%
HT MCS set bitmask	1.49	1.43	1.16	97.6%	99.0%	99.9%	90.9%	90.0%	81.1%
Supported rates	1.18	2.10	1.36	98.2%	95.9%	99.8%	100%	99.9%	100%
Interworking - access net. type	1.08	1.11	0.006	99.6%	99.6%	100.0%	47.5%	46.1%	0.04%
Extended supported rates	1.00	1.77	0.886	98.0%	96.3%	99.4%	99.1%	72.6%	99.7%
WPS UUID	0.878	0.788	0.658	98.2%	99.2%	99.6%	8.4%	5.5%	3.6%
HT extended capabilities	0.654	0.623	0.779	97.8%	98.9%	99.9%	90.9%	90.0%	81.1%
HT TxBeam Forming Cap.	0.598	0.587	0.712	97.8%	98.9%	99.9%	90.9%	90.0%	81.1%
HT Antenna Selection Cap.	0.579	0.576	0.711	98.0%	98.9%	99.9%	90.9%	90.0%	81.1%
Overall	5.48	7.03	5.65	92.5%	90.7%	88.8%	-	-	-

3.1.1 Entropy

We evaluate the quantity of information brought by these different elements using the three datasets introduced in Section 2.4. Following the approach of Panopticlick [16], we empirically evaluate the amount of information provided by each element by computing its entropy in the datasets. The entropy of an element i is computed as follows:

$$H_i = - \sum_{j \in E_i} f_{i,j} * \log f_{i,j} \qquad (2)$$

where E_i is the domain of possible values for element i and $f_{i,j}$ is the frequency (i.e., probability) of the value j for the element i in the dataset. We consider the absence of an element as a possible value.

Results of our analysis of the IEs are presented in Table 2. The *Entropy* column presents the amount of identifying bits provided by the elements. The *Stability* column presents the fraction of devices for which the value of the element is constant throughout the datasets. Finally, the *Affected Devices* column presents the fraction of devices that include this IE in their probe requests.

What appears in this table is that all of these elements are stable for most devices over the observation period. Since most of these IEs reflect intrinsic capabilities of the device, there is no reason for them to change over time. Upon further inspection, it appears that elements which are not stable over time are generated by a small group of device. Most of the studied IEs are present in almost all devices. For instance, the HT capabilities tag, used to advertise capabilities for the High-Throughput 802.11n standard, is the most useful one for fingerprinting. This tag includes a lot of subfields whose values vary from one device to another, providing a lot of identifying information.

There is a high diversity in the amount of information provided by the selected elements. For instance, the HT capabilities info provides up to 4.74 bits of entropy, while the HT Antenna Selection Capabilities provides only 0.711 bit in the best case. This difference can be explained by a larger element (in term of bits), and also by a variance of the value of this element.

Some differences between the datasets are likely due to their age. In particular, some features were not yet widespread when the Sapienza dataset was produced in 2013. Back then, few devices had an Extended Capabilities IE, while now it is wide-spread. Apart from this, the three datasets display the same trends for all the elements.

The *Overall* row presents the information for all the IEs considered together. We can observe that for 88.8% to 93.8% of devices, the included IEs as well as their values do not change over time. More importantly, the amount of information brought by all the IEs together is above 5.4 bits in all three datasets.

Note that the WPS element is not stable for all devices. This does not mean that its content varies over time, but that it is intermittently included by some devices, since we consider the lack of an element as a possible value. When the WPS element is present, it always has the same content.

3.1.2 Anonymity sets

To further study the impact of those IEs, we evaluate the usefulness of the IEs as a device identifier. For each IE fingerprint, we form a set of all the devices sharing this fingerprint (called an anonymity set) and compute the size of this set. Figure 3 shows the distribution of the set sizes. The three datasets exhibit a similar distribution. First, we can observe that there is a significant number of devices alone in their set (leftmost impulse), which means that they have a unique fingerprint. Then, there is a large number of small groups, meaning that although those devices cannot be uniquely identified by the IE fingerprint, they are in a small anonymity set. Finally, there is a small number of large sets, meaning that a large number of devices share the same fingerprint.

This last case is likely caused by highly popular device models: they are found in large numbers and share the same characteristics. A corollary of this observation is that the identifying potential of IEs is reduced for such device models.

Those results show that the IEs can serve as a unique identifier for some devices and that, for the rest of the devices, it can be used as a first step toward full identification.
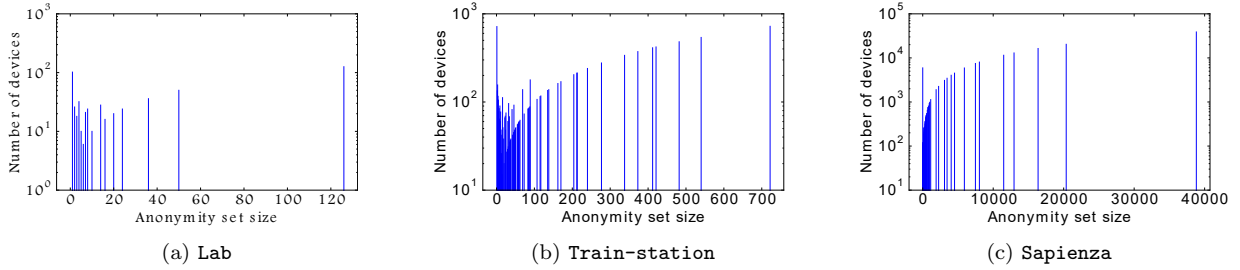
Figure 3: Number of devices that share the same IE fingerprint with a group (i.e., anonymity set) of varying size.

Algorithm 1: WPS UUID generation in wpa_supplicant

Input: *MAC*: MAC address of an interface
Returns: 16-byte WPS UUID

$salt \leftarrow$ `0x526480f8c99b4be5a65558ed5f5d6084`
$UUID \leftarrow$ SHA-1($MAC, salt$)
$UUID[6] \leftarrow (5 \ll 4) \mid (UUID[6]$ & `0x0f`)
$UUID[8] \leftarrow$ `0x80` $\mid (UUID[8]$ & `0x3f`)

return $UUID[:16]$

Table 3: Results of the WPS UUID re-identification attack

Dataset	Number of clients with WPS a tag	Fraction of successfully reversed UUID
`Lab`	8.4%	76.1% (35/46)
`Train-station`	5.5%	73.9% (391/529)
`Sapienza`	3.6%	92.0% (5378/5844)

3.2 Wi-Fi Protected Setup (WPS)

One of the IEs found in probe requests is dedicated to Wi-Fi Protected Setup (WPS), a protocol simplifying device pairing. We show that the unique identifier contained in this IE can be used to reveal the real MAC address of the device.

Some devices add a WPS IE to their broadcast probe requests to advertise their support of the protocol (see Table 3). In our datasets, between 3.7% and 8.6% of devices broadcast at least one probe request with such an IE. One notable field of this IE is the Universally Unique Identifier (UUID) of the device, which is by definition identifying.

There is no official specification for the generation of the UUID, but the Wi-Fi Alliance recommends [47, §3.19] to follow the specification of RFC 4122 [34] and to derive it from the MAC address of one of the device's interfaces. More specifically, RFC 4122 specifies that the UUID should be derived from the truncation of the digest obtained from a cryptographic hashing of the MAC address.

On Linux, `wpa_supplicant` is responsible for the addition of the WPS element. It generates the UUID by computing the SHA-1 hash of the MAC address with a fixed seed, before truncating it. The full algorithm is shown in Algorithm 1. It was shown in [14] that hashed MAC address are reversible through brute-forcing, due to their relatively small address space. Hence it is possible to recover the MAC address that was used to generate the UUID. In other words, if the UUID is calculated in this manner, it leaks the real MAC address.

We calculated the UUID based on the MAC address as described in Algorithm 1 for the `Train-station` and `Lab` datasets. This revealed that roughly 75% of all devices using the WPS IE indeed derive the UUID from the MAC address (see Table 3). For the `Sapienza` dataset, which preserves only the OUI part of the MAC addresses, we attempted to recover the original MAC address by testing all possible values for the last three bytes of the address (together with the given OUI). This proved extremely successful, as this yielded a result for 92% of the devices. Because

we do not have access to the original MAC addresses, we cannot guarantee that all of the recovered addresses are the one used as the Wi-Fi MAC address. Indeed, RFC 4122 [34] recommends to use the address of one of the interfaces, meaning other MAC addresses, such as the Bluetooth one, can be used. We informed the authors of the `Sapienza` dataset about theses de-anonymization issues. Using the same method, we tested our own datasets again, this time exhaustively testing all possible values for the last three bytes of the MAC address, while keeping the advertised OUI. This uncovered 7 new MAC addresses for the `Train-station` dataset, and none for `Lab`. These 7 addresses are all one bit away from the Wi-Fi MAC address of the device, indicating that they are the address of another interface (e.g., the Bluetooth address). We also found a few devices using bogus UUIDs (`12:34:56...` or `00:00:00...`). We conclude that, at the exception of devices using bogus UUIDs, the WPS element is a unique identifier in all our datasets. Moreover, the UUID field of the WPS element can be used to reveal the real MAC address of a device.

3.3 SSID fingerprint

Probe requests include a Service Set Identifier (SSID) element, which is used to specify a network searched by the device. We show that the SSID fingerprint, i.e., the list of SSIDs searched by a device, can be a unique identifier. Devices including this element send multiple probe requests to cover all the SSIDs in their preferred network list (one probe for each network). During each scan iteration, devices send an ordered burst of probe requests over a small timeframe.

Although the practice of putting SSIDs in probe requests is progressively abandoned for obvious privacy reasons, it is still observed for a number of reasons. First, some active devices are not up-to-date and are still running an OS that does not include this privacy-enhancing modification. Second, using a probe request with an SSID is the only way to discover a *hidden* access point. No matter how up-to-date the OS is, a device with configured hidden networks will broadcast the corresponding SSID. Finally, we have observed that some
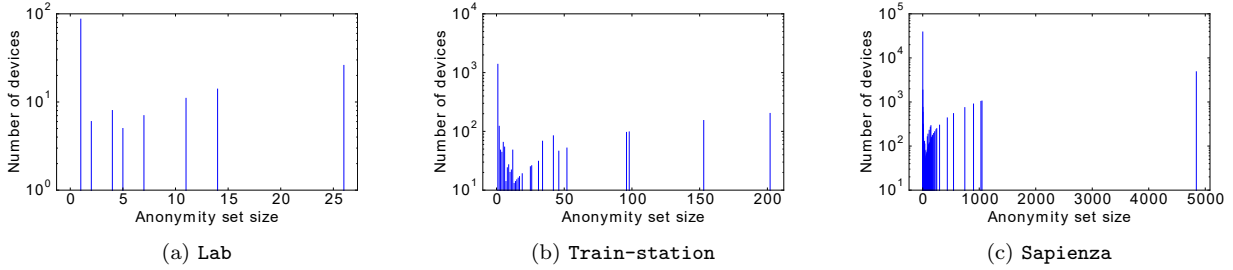
Figure 4: Number of devices that share the same SSID fingerprint with a group (i.e., anonymity set) of varying size.

recent devices like the iPad 2 running iOS 9.1 or the One Plus One running Android 5.1.1 broadcast probe requests with SSIDs when waking up from sleep mode. We conjecture this is because some OSes, as a way to speed up the network-reactivation process, offer separate APIs to initiate background and on-demand (wake up) scans.

In our datasets we found that 29.9% to 36.4% of devices broadcast at least one SSID. Among these, 53% to 64.8% broadcast a unique list of SSIDs. Therefore, this list can be used as an additional unique identifier to track devices.

Using the same method as for IEs, we computed the distribution of anonymity sets for SSIDs. The results are shown in Fig. 4. For readability, we removed the empty SSID list, corresponding to devices which do not broadcast any SSID.

As for IE fingerprints, the three datasets exhibit a similar distribution. For instance, in the Lab dataset, 87 SSID fingerprints are unique, and 26 devices share the same fingerprint. Apart from these extreme values, it appears that the anonymity set of devices sending SSIDs is small ($< 2\%$ of devices). This makes the SSID fingerprint a good tool for identifying and tracking devices.

4. IDENTIFIER-FREE TRACKING

In this section, we present an algorithm to track devices even if MAC address randomization is used. That is, we assume no unique identifiers are available. Our algorithm first clusters probe requests by their Information Element (IE) fingerprint, and then distinguishes devices in each cluster by relying on predictable sequence numbers.

4.1 Adversary and System Model

We assume the adversary is a passive observer who wants to track the movements of people in a certain area. This is done by tracking people's mobile devices, and by placing radio receivers that cover the complete target area. The radios only have to be able to receive broadcast probe requests, full monitor mode support is not required. In practice many institutions, e.g., shopping centers, universities, etc., can use existing infrastructure for this purpose. We assume not all probe requests are captured due to packet loss, and do not require all channels to be monitored. In other words, our algorithm can handle missed packets, and works as long as several consecutive network scans of a device are not missed.

Our algorithm relies on the IE fingerprint and on the predictable sequence numbers of probe requests. Note that all 802.11 frames, apart from control frames, contain a 12-bit sequence number. It is used to detect retransmissions and reconstruct fragmented packets. Based on our tests, all Wi-Fi

radios use an incremental counter to initialize the sequence number. Even when MAC address randomization is enabled, we found that iOS, Linux, and Windows, all use incremental sequence numbers in probe requests. This confirms and extends the observations by Freudiger [18]. Unsurprisingly, in our datasets all devices use an incremental sequence counter. However, roughly one third of devices reset their sequence number on specific occasions. In particular, many devices reset their sequence counter between scan iterations, likely because they turn off the radio chip when idle.

4.2 Tracking Algorithm

Our algorithm works in two phases. First it uses the IE fingerprint to group probes requests into clusters. Then, it relies on predictable sequence numbers to distinguish probe requests sent from different devices within one cluster. If successful, each final cluster corresponds to a unique device.

The full algorithm is shown in Algorithm 2. Its input is the list of probe requests $\mathcal{P}$, and the parameters ΔT and ΔS. Parameter ΔS is the (assumed) maximum distance between sequence numbers of probe requests sent by the same device, and ΔT the (assumed) maximum time between two network scans of a device. The list of probes requests $\mathcal{P}$ can come from multiple APs. The first phase corresponds to the first `forall` loop. In this loop, all probe requests are assigned to some cluster $\mathcal{C}$ based on their IE fingerprint. The algorithm uses the `fingerprint` function to extract the IE fingerprint based on the information elements that are present (see Section 3). The hashmap $\mathcal{M}$ maps fingerprints to the cluster that contains probe requests with the given fingerprint.

In the second phase, our algorithm iterates over all clusters $\mathcal{C}$ in the hashmap $\mathcal{M}$. Here, we rely on sequence numbers and packet arrival times to distinguish devices that have the same IE fingerprint. Effectively, each cluster $\mathcal{C}$ is divided into a list of subclusters $\mathcal{S}$. The sequence number of a probe request is denoted by $p.seq$, and the arrival time by $p.time$. The notation $\mathcal{S}[i].last$ references the last probe request that has been added to the subcluster $\mathcal{S}[i]$. In the nested `forall` loop, we search for a cluster such that the last probe request added to this cluster has an arrival time and sequence number that indicate that it was sent by the same device as probe request p. Care must be taken so devices that reset their sequence number after one scan iteration do not get split up into different clusters. As a heuristic, we assume that if a device exhibits this behaviour, all devices with the same IE fingerprint also have this behaviour. We can then detect devices that reset their sequence number by calculating the maximum sequence number within a cluster. If this number is lower than 100, we assume devices with this fin-

Algorithm 2: Cluster probe requests based on their IE fingerprint and sequence numbers.

Input: $\mathcal{P}$: List of captured probe requests
$\quad\quad\quad\Delta T$: maximum time between two probes
$\quad\quad\quad\Delta S$: maximum sequence number distance
Returns: Set of clusters corresponding to devices

$\mathcal{M} \leftarrow \emptyset \quad\quad$ // $\mathcal{M}$ maps fingerprints to clusters
forall $p \in \mathcal{P}$ **do**
$\quad f \leftarrow \mathrm{fingerprint}(p) \quad$ // Calculate IE fingerprint
$\quad \mathcal{M}[f].\mathrm{append}(p) \quad\quad$ // Append probe to cluster

$\mathfrak{D} \leftarrow [\,] \;$ // List of clusters representing devices
forall $\mathcal{C} \in \mathcal{M}$ **do**
$\quad \mathcal{S} \leftarrow [\,] \quad\quad\quad$ // Will contain subdivision of $\mathcal{C}$
$\quad m \leftarrow \max(p.seq \text{ for } p \text{ in } \mathcal{C})$
$\quad$ **forall** $p \in \mathcal{C}$ **do**
$\quad\quad$ Find i such that: $\quad$ // Find matching cluster
$\quad\quad\quad \mathrm{d}(\mathcal{S}[i].last.seq,\, p.seq, m) \le \Delta S$
$\quad\quad\quad$ and $p.time - \mathcal{S}[i].last.time \le \Delta T$
$\quad\quad$ **if** *no i found* **then**
$\quad\quad\quad i \leftarrow |\mathcal{S}| \quad\quad\quad$ // Create new subcluster
$\quad\quad \mathcal{S}[i].\mathrm{append}(p) \quad\quad$ // Add p to subcluster
$\quad \mathfrak{D}.\mathrm{extend}(\mathcal{S}) \quad\quad$ // Extend list $\mathfrak{D}$ with $\mathcal{S}$
return $\mathfrak{D}$

---

gerprint reset their sequence number. We get the following definition for the distance between two sequence numbers:

DEFINITION 3. *The* sequence distance $d(x, y, max)$ *between two sequence numbers x and y is defined as:*

$$d(x, y, max) = \begin{cases} |x - y| & \text{if } max < 100 \\ y - x & \text{if } x < y \\ 2^{12} - x + y & \text{otherwise} \end{cases}$$

Here max represents the maximum sequence number in a given cluster. All subclusters are appended to the final list of subclusters $\mathfrak{D}$. Finally, the algorithm returns $\mathfrak{D}$, where it is assumed each cluster in $\mathfrak{D}$ corresponds to one device.

4.3 Evaluation

We investigated the performance of our algorithm based on our real-world datasets. To control the number of concurrent devices, and the duration that they are present, we first filtered these datasets. To only simulate devices that remain in the tracked area for a given duration, we removed devices of which we lost too many consecutive probe requests. This indicates that the device moved outside the tracked area. We rely on sequence numbers to determine how many frames are lost: if 64 or more consecutive frames are lost, we assume the device moved outside the tracked area, and we remove the device. For devices that reset their sequence number after each scan iteration, we only base ourselves on the time between frames to determine if a device went out of range. We also removed the WPS information element in all probe requests, and replaced all SSIDs with a broadcast (empty) SSID. This assures we are tracking devices without relying on obvious unique identifiers. We only make use of MAC addresses to measure the performance of our algorithm.

We consider a device to be successfully tracked if there is exactly one cluster that contains all probe requests sent
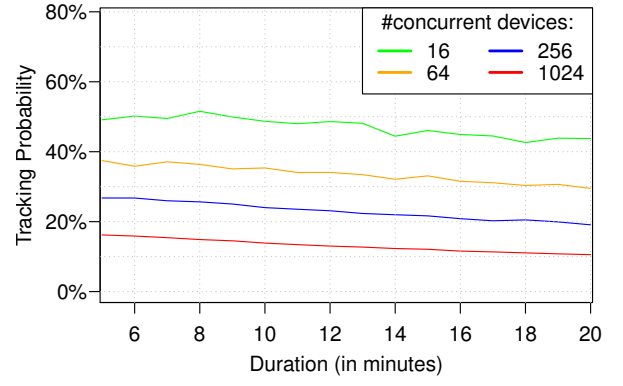

Figure 5: Probability of a device being successfully tracked using Algorithm 2, in function of the duration that the device was present, and the number of concurrent devices.

by this device, and no other frames are in this cluster. Put differently, all probe requests of this device have to be successfully linked together without a single error. With this definition, the tracking probability under various conditions is shown in Fig. 5. We used a value of 64 for ΔS, and 500 seconds for ΔT. These rather large values are picked so the tracking algorithm can tolerate several missed probe requests. Our results are promising. Even when simulating as much as 1024 concurrent devices, over a duration of 20 minutes, we manage to successfully track a significant amount of devices. For shorter tracking durations, and when the number on concurrent devices is more realistic, we manage to track roughly half of all devices.

4.4 Discussion and Countermeasures

The main reason why certain devices are not successfully tracked, is because some clusters contain probe requests of multiple devices. In Section 5, we show that scrambler seeds can further distinguish devices in these clusters. The second type of error is that probe requests of some devices are spread out over multiple clusters. This is caused by the variability of the IE fingerprint (see Section 3). Hence, improvements to the fingerprint function may further increase the tracking probability of our algorithm.

In our datasets, we generally only monitor one channel. This makes it harder to distinguish devices using sequence numbers, since the average gap between sequence numbers of captured frames is relatively high. Monitoring multiple channels may further increase the tracking probability.

The 802.11 standard only requires that the same sequence number is used for retransmissions, and that the same number is used for all fragments of a packet [31, §8.2.4.4.2]. Hence, one can reset the sequence counter to a random (unused) value if a new MAC address is being used.

5. PREDICTABLE SCRAMBLER SEEDS

In this section, we study the scrambler seeds of commodity Wi-Fi radios, and find that all of them use predictable seeds. We show this can be used to improve our tracking algorithm.

5.1 Background and Experimental Setup

Recently, Bloessl et al. discovered that the scrambler seeds of two (prototype) radios used in wireless vehicular networks are predictable [6]. They showed this can be used to improve

vehicle tracking algorithms. While the 802.11 standard says that scrambler seeds should be initialized with a pseudo-random nonzero seed [31, §18.3.5.5], we wondered whether commodity Wi-Fi radios also use predictable seeds in practice. To answer this question, we need a radio that exports the scrambler seed of received Wi-Fi frames. Since most commodity devices do not do so, we implemented this ourselves using a software-defined radio. We used an Ettus USRP N210, and relied on the `gr-ieee802-11` project [5] to decode OFDM frames. The code was modified to take the scrambler initialization value from the SERVICE field, and undo the initial 7 shifts to obtain the original scrambler seed value (see Section 2.2).

Because `gr-ieee802-11` is not as optimized as real Wi-Fi receivers, decoding frames using it is not easy. To increase its reliability, all captures were made in an RF-shielded room. For each device being tested, we made it transmit data frames of various lengths, and using different bitrates. Based on these captures, we studied the predictability of the scrambler seed. In our analysis, we mainly focus on the scrambler seed behaviour of a device when it is transmitting frames at 6 Mbps. This is done because probe requests in the 5 GHz band are always sent at a bitrate of 6 Mbps (see Section 2.2). Finally, we confirmed our predictions by capturing and analyzing real probe requests in the 5 GHz band.

5.2 Analysis

We found that most devices do not reset the state of the scrambler at all. Put differently, the state of the LFSR after transmitting a frame is reused as the seed of the next frame. We say these LFSRs are used in a *free-wheeling* mode, where the state is never explicitly initialized. Let *end state* denote the state of the LFSR after producing the last bit of the scrambler sequence. Then one would expect that the end state is directly used as the seed for the next frame. Interestingly, we found that most devices perform additional LFSR shifts before writing out the next scrambler seed. It is unclear why devices do this, perhaps for alignment reasons. Nevertheless, in our case, it is only important to predict how many additional LFSR shifts are performed to get the scrambler seed. To rigorously analyze this behaviour, we define the *shift distance* between two LFSR states as follows:

DEFINITION 4. *The* shift distance $D_L(x,y)$ *between two LFSR states x and y is defined as:*

$$D_L(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 + D_L(S_L(x),y) & \text{otherwise} \end{cases}$$

Recall that $S_L(x)$ represents the result of one LFSR shift on the state x. Hence, the shift distance is the number of shifts needed to reach the second state from the first state. The shift distance allows us to report how many additional shifts a device performs before writing out the seed value into the SERVICE field. If we state that a device uses a particular shift distance, it implies it operates in a free-wheeling mode, and the reported distance denotes the shift distance between the *end state* and the scrambler seed of the next frame.

5.2.1 Asus Fonepad (K004 ME371MG)

This radio always uses a shift distance of 22, making it trivial to predict the next scrambler seed value based on the previous frame.

Table 4: Intel 7260 AC shift distances in function of the bitrate and PSDU length (in bytes) of the previous frame.

Bitrate	PSDU byte length modulo 12											
	0	1	2	3	4	5	6	7	8	9	10	11
6 Mbps	22	46	14	14	54	54	22	46	46	46	22	22
12 Mbps	54	54	54	54	54	38	54	54	54	38	38	38

5.2.2 One Plus and Samsung Galaxy A3

The radio in these devices always uses a shift distance of 6, once again making it trivial to predict the next seed value.

5.2.3 TP-Link TL-WN821N

This device uses an RTL8192CU radio chip which always uses a fixed seed value of 124. We consider this a bug in the radio: using a fixed scrambler seed value means frames are always mapped to the same physical signal. If this happens to be a disadvantageous signal, for example because it has a high peak to average power ratio [44], retransmissions are also sent using this disadvantageous signal. Hence, certain frames experience systematically higher frame error rates [6].

Nevertheless, even a fixed seed value can be used to improve our tracking algorithm. For example, when tracking a device with a fixed seed, we can exclude all frames with a different seed value as coming from this device. Without access to the seed, we could have incorrectly labelled certain frames as being transmitted by this particular device.

5.2.4 iPad Air 2 (A1566)

We found that this device only uses seed values of 8, 64, and 72. The seed value 72 was used considerably more than 8 and 64. Interestingly, in these three values, only the 4th and 7th bits are ever set. These bit positions correspond to the LFSR feedback taps used in the scrambler (see Fig. 1). Similar to the TP-Link device, we consider using only a few fixed values for the scrambler seed to be a bug in the radio.

5.2.5 Wi-Pi: Ralink RT5370

This radio also operates in a free-wheeling mode. After transmitting a frame using a bitrate of 6 Mbps and 12 Mbps, it uses a shift distance of 6. After transmitting a frame using a bitrate of 9 Mbps, it uses a shift distance of 10. Somewhat surprisingly, it uses a shift distance of 61 after sending a beacon at 6 Mbps. This serves as a good example that for some devices, the shift distance may also depend on the type of the previous frame. Thankfully, probe requests are always sent at 6 Mbps in the 5 GHz band, and no other frames are transmitted in between. Consequently, the scrambler seed of radios exhibiting this behaviour remains predictable.

5.2.6 Intel 7260 AC

This radio operates in a free-wheeling mode, where the shift distance depends on the bitrate and the PSDU length of the previously transmitted frame. Recall that the PSDU denotes the actual MAC layer bytes being transmitted (see Fig. 2). To better analyze this behaviour, we also made several captures were we sent frames at 12 Mbps. The resulting shift distances are summarized in Table 4. For example, if the previous frame was sent at 6 Mbps, and had a PSDU length of 13 bytes, the shift distance will be 46.

5.2.7 Samsung Galaxy S3 & iPhone 5 (A1429)

The radios in these devices all exhibit the same behaviour, and operate in a free-wheeling mode. When the previous frame was sent at 6 Mbps, it uses a shift distance from the set $\{110, 114, 118, 122\}$. We are currently unable to predict which shift value in this set is used. When the previous frame was sent at 12 Mbps, the set of possible shift distances is $\{102, 106, 110, 114\}$. While this means we cannot precisely predict the next scrambler seed value, it still can be used to improve the tracking probability. We conjecture that a more detailed study of this radio chip, e.g., reverse engineering its operations, would reveal how a shift distance is selected from our current set of possible distances.

5.2.8 Atheros AR9271

The Atheros AR9271 radio uses an incremental counter to initialize the scrambler seed. That is, the scrambler seed is explicitly initialized, and incremented by one for each transmitted frame. This makes it easy to predict the next seed value, even when some frames of unknown lengths are missed. It is the only device we tested that does not operate in a free-wheeling mode. We confirmed this behaviour using an Alfa AWUS036NHA and a TP-Link TL-WN722N, which both contain an Atheros AR9271 radio.

5.3 Improved Tracking Algorithm

The scrambler bit provides us 7 extra bits of information to identify a device. That is, in the tracking algorithm of section 4, we can now distinguish devices based on the scrambler seeds of frames. Though current Wi-Fi radios do not export the received seed, we believe it is easy for manufacturers to support this. In fact, newer radios may already be doing this to support the 802.11ac standard. In 802.11ac, certain bits of the seed in RST and CTS frames have a special purpose [20]. This requires that certain bits of the seed must remain available after demodulating the physical signal. This makes it more likely that future devices can, and perhaps will, export the scrambler seed of received frames.

In practice, we must be able to determine the type of scrambler a device uses. Otherwise, we cannot predict the next scrambler seed. Since all devices use the same (random) MAC address in one scan iteration, we can easily determine the type of scrambler used by grouping the frames based on the MAC address. Another option is to immediately send a probe reply when receiving a probe requests. In turn, the device will send an ACK, which will also contain a scrambler seed. Based on the probe request and the ACK, we can determine the type of scrambler being used. Hence, in our tracking algorithm, we can assume the type of scrambler is known.

To simulate knowledge of scrambler seeds, the first frame of each device is assigned a random seed value. Since only probe requests sent in the 5 GHz band contain a scrambler seed, we expect that few scrambler seed values will be missed by an attacker. Hence, we assign subsequent frames a random seed that lies within a distance of 16 or less of the previous frame. Similarly, in Algorithm 2, we search a cluster such that the last probe request added to this cluster has a scrambler seed that is within a distance of 16 of the probe requests being added. Figure 6 shows the impact on the tracking probability when knowledge of scrambler seeds is simulated in this manner. For comparison, the tracking probability without using scrambler seeds is shown as
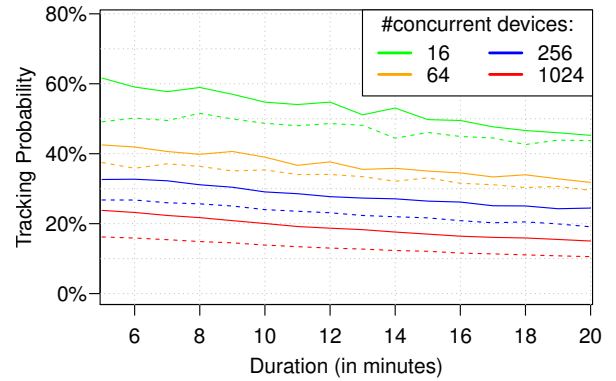

Figure 6: Probability of a device being successfully tracked, in function of the duration that the device was present, and the number of concurrent devices. Tracking is done using Algorithm 2, with the addition that knowledge of scrambler seeds is also simulated. The dashed line is the probability as reported in Fig. 5, and is repeated for convenience.

a dashed line. We conclude that using scrambler seeds can increase the tracking probability by as much as 10%.

6. FAKE ACCESS POINTS ATTACKS

In this Section we show how two service discovery mechanisms of Wi-Fi can be abused to gain identifying information on unassociated stations. This is accomplished by creating fake APs with specific characteristics and identifiers.

6.1 Reviving the Karma Attack

Once a device has detected an AP advertising an SSID matching one of its preferred (configured) networks, it will automatically initiate the association process with this AP. From this point on, most devices that implement MAC address randomization will use their real MAC address to connect with the AP. The only exception is Windows, which uses a per-network random MAC address (see Section 2.1). Using the real MAC address may be necessary because some APs restrict association and Internet connectivity based on the MAC address being used.

A consequence of this switch to the real MAC address when associating to an AP is that, by advertising an SSID familiar to a device, the latter will automatically reveal a permanent identifier. Existing attacks that advertised SSIDs in order to get association requests from victims, such as the well-known Karma attack [11], relied on the SSIDs that the victim broadcasts in probe requests. However, since modern devices avoid broadcasting SSIDs, the Karma attack is no longer applicable. Our solution to this problem is to advertise a list of popular SSIDs, hoping that at least one of them is in the preferred network list of the victim.

As seen in section 3.3, popular SSIDs are configured in a large number of devices. Indeed, SSIDs configured in Wi-Fi devices follow a long-tailed distribution, which means that a small number of popular SSIDs are found in many devices.

Using our datasets, we simulate the efficiency of this attack. We compute the number of devices that include at least one of the top-n popular SSIDs. Figure 7 presents the cumulative distribution of the number of affected devices in function of the number of advertised SSIDs. We can see that a relatively short list of popular SSIDs is enough to
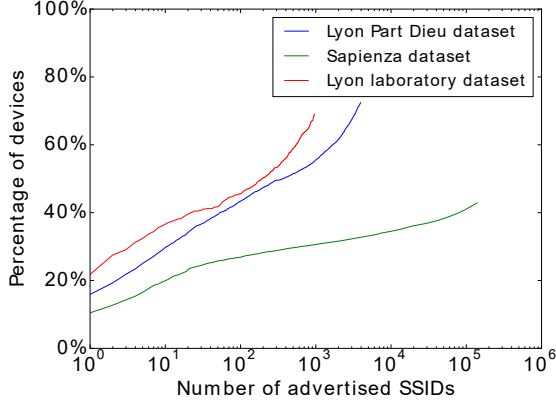
Figure 7: Cumulative distribution of the number of affected devices as a function of the number of SSIDs.

trigger association requests from a large fraction of the devices. For instance, the top-20 SSIDs in the `Train-station` dataset covers more than 33 % of the devices.

We tested and verified this attack in practice. `Airbase-ng` was used to create fake APs from a list of popular SSIDs. We made the assumption that the most popular SSIDs are open hotspots that do not use encryption. Therefore, the APs we broadcast were configured to be open hotspots. Although `airbase-ng` supports the creation of multiple SSIDs, our experiments revealed that it does not properly handle a large numbers of SSIDs. Therefore, we limited our attack to a reasonable number of SSIDs, i.e., 5 SSIDs. This number of advertised SSIDs can be increased by running multiple instances of `airbase-ng` on distinct interfaces.

The performance of the attack is evaluated by computing the fraction of devices that are trying to associate (connect) to our fake AP over the total number of devices that have been observed. A device is considered as *observed* when we collected at least 5 probe requests from its MAC address. This conservative approach was taken to avoid counting devices that are too far to detect our AP, or that are not staying in range long enough to start the association process.

We executed the attack for one hour using a list of the 5 top SSIDs of the `Train-station` dataset in a dense urban area. During this experiment, a total of 2481 stations were observed and our fake APs triggered association attempts from a total of 434 devices (17.4%). Increasing the number of advertised SSIDs, as well as responding to probe requests containing an SSID with a matching probe response, has the potential to improve the number of affected devices.

Note that this attack does not necessarily need to be active. An attacker could rely on nearby, genuine, APs with popular SSIDs to trigger association requests. In many locations, APs with popular SSIDs are commonplace. For instance, in Europe, Wi-Fi networks such as `Fon` or `Free_WiFi` can be found in large numbers in populated areas.

6.2 Hotspot 2.0 Honeypot

We show that the service discovery mechanisms introduced by Hotspot 2.0 (HS2.0) can be leveraged to reveal the real MAC address of Linux and Windows devices. Additionally, we discover another predictable counter that can be used to improve our tracking algorithm of Section 4.

Because HS2.0 APs do not advertise all their capabilities in beacons, stations need to retrieve the full list by contacting the AP. In practice, a station sends ANQP queries to the AP, and the latter replies with an ANQP response containing a list of the available services (see Section 2.3).

The real identity of devices can be easily uncovered if stations use their real MAC address for querying the AP. Our observations confirm that this is the case for a Linux computer using `wpa_supplicant`, and a computer running Windows 10. However, an Apple device running iOS 9.1 kept its randomized MAC address when sending the ANQP query. We found that Windows and iOS devices only send ANQP queries if at least one HS2.0 network was configured on the device.

The fact that the iOS device does not fall back to a persistent identifier is a positive point. However, ANQP queries include a 1-byte field named `Dialog Token` that is used to identify each GAS transaction [32, §7.4.7.13]. As multiple GAS transactions are initiated, the value of this field will be incremented. The `Dialog Token` field is therefore predictable and could be used in our tracking algorithm in the same way sequence numbers and scrambler seeds were used (see Section 4 and 5, respectively). In our captures, the typical time between two consecutive queries is below 60 seconds, providing a good temporal granularity for tracking.

A variant of the Fake AP attack can therefore be mounted by leveraging this service discovery mechanism: a simple AP advertising HS2.0 capabilities will induce all surrounding HS2.0 compatible stations to send ANQP queries, thus revealing their real MAC address or a predictable field.

We tested this attack in practice using `hostapd`, which was configured to advertise HS2.0 capabilities. A fake HS2.0 AP was deployed in a dense urban location for two 20 minute sessions. During those sessions, 1523, respectively 562, probing stations were observed. At the same time, we recorded 80 (5.25%), respectively 92 (16.37%) stations sending ANQP queries to the HS2.0 AP. This small fraction can be explained by the fact that this technology is still in the early stages of its deployment. Hence, not all devices support this feature. Additionally, as HS2.0 is not yet widespread, only a few devices have a configured HS2.0 network. This means iOS and Windows devices will not send ANQP queries.

Finally, we observed that none of the stations sending ANQP queries had a MAC address that was registered by Apple. This confirms the previous observations about the requirements of at least one configured HS2.0 network to get involved in HS2.0 service discovery requests.

Our results show that the recent HS2.0 standard is potentially a source of privacy leakage, since several implementations reveal their real MAC address or a predictable field when initiating ANQP queries. The importance of this issue will increase with the number of compatible stations, but also with the increasing number of access points. Indeed, similar to the previous attack, pervasive deployment of genuine HS2.0 will remove the need of creating a fake AP with HS2.0 capabilities. We hope that by discovering this attack in the early stages of the adoption of HS2.0, manufacturers will address this issue before HS2.0 becomes widespread.

A simple countermeasure is to follow the iOS example, by using the temporary random MAC address when querying HS2.0. There is indeed no requirement in the specifications [32, §5.9] for the station to use its real MAC address

when querying list of services to an AP. Similarly, picking a cryptographic random value for the `Dialog Token` assures it cannot be used to track devices.

7. RELATED WORK

The possibility to track individuals based on the radio signals of their mobile device has received considerable attention from the research community. Musa et al. [37] used Wi-Fi tracking techniques to collect data on urban mobility. Surveillance systems based on Wi-Fi tracking have also been presented by several works [30, 13, 38]. Cuthbert et al. demonstrated how an airborne Wi-Fi sniffer can be used to passively track and locate owners of Wi-Fi devices [13]. Humbert et al. studied tracking strategies against mobile users that change identifiers when entering mix-zones [29].

Privacy issues of Wi-Fi-enabled devices were studied by Greenstein et al. [23]. In parallel, attempts were made at reducing private information leakage in Wi-Fi [25, 40, 35]. Gruteser et al. introduced the idea of using a disposable identifier instead of a permanent MAC address [25]. Then, proposals [40, 35] were made to improve the privacy in the service discovery mechanisms of Wi-Fi. These improvements involve the obfuscation of device and network identifiers and require significant modifications of the protocol.

After the introduction of MAC address randomization in iOS 8, several works have attempted to understand its internals and started to identify limitations [36, 18]. In particular, Freudiger found that sequence numbers and timing information can be used to re-identify random MAC addresses as implemented by iOS [18]. However, they did not investigate other devices or operating systems, nor other means to de-anonymize or track Wi-Fi frames.

Bernados et al. [8] studied the feasibility of MAC address randomization for associated devices in real-life conditions. They found that existing devices can support this with only minor changes, but note that higher layers must also be configured so they do not leak any identifiers.

Bloessl et al. showed that the scrambler seed of the IEEE 802.11p physical layer could be used for tracking in wireless vehicular networks [6]. They specifically investigated two prototype radio chips. The first one is a radio chip implemented on Field Programmable Gate Array (FPGA), and the second is an industrial-grade Atheros AR5414A-B2B chip. Both were using predictable scrambler seeds.

Others focused on inter-frame arrival time and sequence numbers to identify devices. Guo et al. [26] use these values to detect spoofing on a network with a semi-active method, as an intrusion detection mechanism. Desmond et al. [15] fingerprint devices using inter-frame time analysis alone. They reach a success rate of 70 to 80% to differentiate frames from 45 different devices probing for more than one hour.

Other means of device fingerprinting can be used to defeat MAC address randomization. Physical layer fingerprinting of wireless devices based on unique characteristics of their hardware is possible [7, 12] but requires expensive dedicated hardware. Pang et al. [39] showed that devices replacing their identifiers with temporary ones are still trackable through traffic analysis. They focused on MAC-layer fields used by associated devices. In [17], Franklin et al. showed that the inter-frame timing of probe requests can form a fingerprint of the wireless device driver. Access points are vulnerable to passive clock skew fingerprinting [33]. This technique was reused by Cristea et al. [10] to fingerprint

smartphones, but it relies on TCP timestamps, which are not sent by unassociated devices. Stöber et al. apply passive traffic analysis to identify devices, even when the traffic is encrypted [43].

8. CONCLUSION

We study data contained in Wi-Fi probe requests, and demonstrate that they hold enough information to perform tracking, even if the MAC address is periodically randomized. First, the list of information elements form a fingerprint of a device. This fingerprint has enough entropy to identify a device or a small group of devices. We show that when combining this fingerprint with frame sequence numbers, tracking devices is possible regardless of their MAC address. In particular we rely on the observation that most devices do not reset the frame sequence counter when the MAC address is changed. In addition, we show that the scrambler seeds used at the physical layer is predictable in many commodity Wi-Fi devices. Being managed at the hardware level, there is currently no way to reset nor change its value, which makes it a persistent threat. Finally, we introduce two active attacks that leverage service discovery mechanisms of Wi-Fi in order to obtain identifying information on devices that are using MAC address randomization.

Our findings highlight the difficulty of implementing anti-tracking solutions for wireless devices. In particular, it shows that MAC address randomization alone is not enough to protect users' privacy. However, several measures can be taken to limit the impact of the issues presented in this paper. At the software level, the driver (or firmware) should reset the sequence number field as well as any other predictable field found in frames. Additionally, the amount of information elements in probe requests should be kept to a bare minimum to avoid fingerprinting. Finally, Wi-Fi radios, which are generally implemented in hardware, should be modified so scrambler seeds are not predictable. This can be accomplished by letting the hardware generate cryptographically random seeds, or by allowing the driver to instruct the radio which scrambler seed value should be used.

9. ACKNOWLEDGMENTS

10. REFERENCES

[1] Tails - privacy for anyone anywhere. Retrieved from https://tails.boum.org.

[2] Android 6.0 changes. Retrieved from https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html, 2015.

[3] O. Abukmail. Wifi Mac Changer. Retrieved from https://play.google.com/store/apps/details?id=com.wireless.macchanger.

[4] M. V. Barbera, A. Epasto, A. Mei, S. Kosta, V. C. Perta, and J. Stefa. CRAWDAD dataset sapienza/probe-requests (v. 2013-09-10). Retrieved 10 November, 2015, from, http://crawdad.org/sapienza/probe-requests/20130910, Sept. 2013.

[5] B. Bloessl, M. Segata, C. Sommer, and F. Dressler. An IEEE 802.11 a/g/p OFDM receiver for GNU radio. In *SRIF Workshop*, 2013.

[6] B. Bloessl, C. Sommer, F. Dressler, and D. Eckhoff. The scrambler attack: A robust physical layer attack on location privacy in vehicular networks. In *ICNC*, 2015.

[7] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless device identification with radiometric signatures. In *MobiCom*, 2008.

[8] P. O. Carlos J. Bernardos, Juan Carlos Zúñiga. Wi-Fi internet connectivity and privacy: hiding your tracks on the wireless internet. In *IEEE CSCN*, 2015.

[9] Chainfire. Pry-Fi. Retrieved from https://play.google.com/store/apps/details?id=eu.chainfire.pryfi.

[10] M. Cristea and B. Groza. Fingerprinting smartphones remotely via ICMP timestamps. *Communications Letters, IEEE*, 17(6):1081–1083, 2013.

[11] D. A. Dai Zovi, S. Macaulay, et al. Attacking automatic wireless network selection. In *Proc. of the Sixth Annual SMC Inf. Assurance Workshop*, 2005.

[12] B. Danev, D. Zanetti, and S. Capkun. On physical-layer identification of wireless devices. *ACM Computing Surveys (CSUR)*, 45(1):6, 2012.

[13] C. Daniel and W. Glenn. Snoopy: Distributed tracking and profiling framework. In *44Con 2012*, 2012.

[14] L. Demir, M. Cunche, and C. Lauradoux. Analysing the privacy policies of Wi-Fi trackers. In *Proc. of the 2014 workshop on physical analytics*, 2014.

[15] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee. Identifying unique devices through wireless fingerprinting. In *WiSec*, 2008.

[16] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, 2010.

[17] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX Security*, 2006.

[18] J. Freudiger. How talkative is your mobile device? An experimental study of Wi-Fi probe requests. In *WiSec*, 2015.

[19] B. Gellman and A. Soltani. NSA tracking cellphone locations worldwide, Snowden documents show. *The Washington Post*, 2013.

[20] M. X. Gong, B. Hart, L. Xia, and R. Want. Channel bounding and MAC protection mechanisms for 802.11ac. In *GLOBECOM*, 2011.

[21] F. Gont. A method for generating semantically opaque interface identifiers with ipv6 stateless address autoconfiguration (slaac). RFC 7217, 2014.

[22] D. Goodin. No, this isn't a scene from minority report. This trash can is stalking you. *Ars Technica*, 2013.

[23] B. Greenstein, R. Gummadi, J. Pang, M. Y. Chen, T. Kohno, S. Seshan, and D. Wetherall. Can Ferris Bueller still have his day off? protecting privacy in the wireless era. In *USENIX HotOS*, 2007.

[24] E. Grumbach. iwlwifi: mvm: support random MAC address for scanning. Linux commit `effd05ac479b`.

[25] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers: A quantitative analysis. *Mobile Networks and Applications*, 10(3):315–325, 2005.

[26] F. Guo and T. Chiueh. Sequence number-based MAC address spoof detection. In *RAID*, 2006.

[27] C. Huitema. Experience with MAC address randomization in Windows 10. In *93th Internet Engineering Task Force Meeting (IETF)*, July 2015.

[28] C. Huitema. Personal communication, Nov. 2015.

[29] M. Humbert, M. H. Manshaei, J. Freudiger, and J.-P. Hubaux. Tracking games in mobile networks. In *Conf. on Decision and Game Theory for Security*, 2010.

[30] N. Husted and S. Myers. Mobile location tracking in metro areas: Malnets and others. In *CCS*, 2010.

[31] IEEE Std 802.11-2012. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[32] IEEE Std 802.11u. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 9: Interworking with External Networks*, 2011.

[33] S. Jana and S. K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. In *MobiCom*, 2008.

[34] P. Leach, M. Mealling, and R. Salz. A universally unique identifier (UUID) URN namespace. RFC 4122 (Proposed Standard), July 2005.

[35] J. Lindqvist, T. Aura, G. Danezis, T. Koponen, A. Myllyniemi, J. Mäki, and M. Roe. Privacy-preserving 802.11 access-point discovery. In *WiSec*, 2009.

[36] B. Misra. iOS 8 MAC randomization – analyzed! http://blog.airtightnetworks.com/ios8-mac-randomization-analyzed/.

[37] A. B. M. Musa and J. Eriksson. Tracking unmodified smartphones using Wi-Fi monitors. In *SenSys*, 2012.

[38] B. O'Connor. CreepyDOL: Cheap, distributed stalking. In *BlackHat*, 2013.

[39] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *MobiCom*, 2007.

[40] J. Pang, B. Greenstein, S. Seshan, and D. Wetherall. Tryst: The case for confidential service discovery. In *HotNets*, 2007.

[41] J. Scahill and G. Greenwald. The NSA's secret role in the U.S. assassination program. *The Intercept*, 2014.

[42] K. Skinner and J. Novak. Privacy and your app. In *Apple Worldwide Dev. Conf. (WWDC)*, June 2015.

[43] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are?: smartphone fingerprinting via application behaviour. In *WiSec*, 2013.

[44] L. Wang and C. Tellambura. An overview of peak-to-average power ratio reduction techniques for OFDM systems. In *IEEE ISSPIT*, 2006.

[45] W. Wang. Wireless networking in Windows 10. In *Windows Hardware Engineering Community conference (WinHEC)*, Mar. 2015.

[46] Wi-Fi Alliance. *Hotspot 2.0 (Release 2) Technical Specification v1.1.0*, 2010.

[47] Wi-Fi Alliance. *Wi-Fi Simple Configuration Protocol and Usability Best Practices for the Wi-Fi Protected Setup Program, v2.0.1*, April 2011.