Jeremy Martin*, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik C. Rye, and Dane Brown

A Study of MAC Address Randomization in Mobile Devices and When it Fails

Abstract: Media Access Control (MAC) address randomization is a privacy technique whereby mobile devices rotate through random hardware addresses in order to prevent observers from singling out their traffic or physical location from other nearby devices. Adoption of this technology, however, has been sporadic and varied across device manufacturers. In this paper, we present the first wide-scale study of MAC address randomization in the wild, including a detailed breakdown of different randomization techniques by operating system, manufacturer, and model of device.

We then identify multiple flaws in these implementations which can be exploited to defeat randomization as performed by existing devices. First, we show that devices commonly make improper use of randomization by sending wireless frames with the true, global address when they should be using a randomized address. We move on to extend the passive identification techniques of Vanhoef et al. to effectively defeat randomization in ∼96% of Android phones. Finally, we identify a previously unknown flaw in the way wireless chipsets handle low-level control frames which applies to 100% of devices we tested. This flaw permits an active attack that can be used under certain circumstances to track any existing wireless device.

Keywords: MAC address, randomization, privacy, tracking, 802.11, WiFi, hardware identifiers

DOI 10.1515/popets-2017-0054
Received 2017-02-28; revised 2017-06-01; accepted 2017-06-02.

***Corresponding Author: Jeremy Martin:** The MITRE Corporation, work done partly while at the US Naval Academy (USNA), E-mail: jbmartin@mitre.org
Travis Mayberry: USNA, E-mail: mayberry@usna.edu
Collin Donahue: USNA
Lucas Foppe: USNA
Lamont Brown: USNA
Chadwick Riggins: USNA
Erik C. Rye: USNA, E-mail: rye@usna.edu
Dane Brown: USNA, E-mail: dabrown@usna.edu

1 Introduction

Smartphones are one of the most impactful technologies of this century. The ability to access the Internet anytime and anywhere has fundamentally changed both work and personal life across the globe [27]. It is gradually becoming clear, however, that in exchange for this level of access to the Internet people may be giving up a substantial amount of privacy. In particular, it has recently been made public that state sponsored intelligence agencies, in countries such as Russia and China [5, 7, 19], as well as private sector companies [22], are actively attempting to track cellphone users.

Smartphones conventionally have two major modes of communication, both of which can potentially be used to track users. The first and most obvious is the cellular radio itself [10, 25]. However, an often overlooked second avenue for tracking cellphones (and their corresponding users) is the 802.11 (WiFi) radio that most smart phones also use.

Every 802.11 radio on a mobile device possesses a 48-bit link-layer MAC address that is a globally unique identifier for that specific device. The MAC address is a crucial part of WiFi communication, being included in every link-layer frame that is sent to or from the device. This unfortunately poses a glaring privacy problem because any third party eavesdropping on nearby WiFi traffic can uniquely identify nearby cellphones, and their traffic, through their MAC addresses [12].

There is one particular type of WiFi packet, called a *probe request frame*, that is an especially vulnerable part of WiFi traffic with respect to surveillance. Since probe requests continuously broadcast at a semi-constant rate they make tracking trivial. Mobile devices are effectively playing an endless game of digital "Marco Polo," but in addition to "Marco" they are also broadcasting out their IDs (in the form of a MAC address) to anyone that cares to listen. To address this problem, some modern mobile devices make use of temporary, randomized MAC addresses that are distinct from their true global address. When probe requests are sent out, they use a randomized *pseudonym* MAC address that is changed periodically. A listener should be unable to continuously

track the phone because the MAC changes in a way that hopefully cannot be linked to the previous address.

In this work we evaluate the effectiveness of various deployed MAC address randomization schemes. We first investigate how exactly different mobile Operating Systems (OSs) actually implement randomization techniques, specifically looking at how the addresses are generated and under what conditions the devices actually use the randomized address instead of the global one. Using real-world datasets we provide the first evaluation of adoption rates for randomization across a diverse manufacturer and model corpus.

After establishing the current state of randomization for widely used phone models and OS versions, we move on to show several weaknesses in these schemes that allow us to track phones within and across multiple collections of WiFi traffic. Our work builds on the fingerprinting techniques of Vanhoef et al. [28] in addition to new approaches for deanonymizing phones based on weaknesses we discovered while analyzing wireless traffic from many randomizing phones. This paper makes the following novel contributions:

- We decompose a large 802.11 corpus, providing the first granular breakdown of real-world MAC address randomization. Specifically, we develop novel techniques to identify and isolate randomization and randomization schemes from large collections of wireless traffic.
- We present the first manufacturer and device breakdown for MAC randomization, describing the particular technique each uses. Our results indicate that adoption rates are surprising low, specifically for Android devices.
- We review previous techniques for determining global MAC addresses and find them to be insufficient. We provide additional context and improvements to existing passive and active techniques, substantially increasing their effectiveness.
- We identify significant flaws in the majority of Android randomization implementations which allow for trivial retrieval of the global MAC address.
- Discovery and implementation of a control frame attack which exposes the global MAC address (and thus allows tracking/surveillance) for all known devices, regardless of OS, manufacturer, device type, or randomization scheme. Furthermore, Android devices can be susceptible to this attack even when the user disables WiFi and/or enables Airplane Mode.
- We propose a set of best practices towards developing a secure MAC address randomization policy.

2 Background

2.1 MAC Addresses

Every network interface on an 802.11 capable device has a 48-bit MAC address layer-2 hardware identifier. MAC addresses are designed to be persistent and globally unique. In order to guarantee the uniqueness of MAC addresses across devices the Institute of Electrical and Electronics Engineers (IEEE) assigns blocks of addresses to organizations in exchange for a fee. A MAC Address Block Large (MA-L), commonly known as an Organizationally Unique Identifier (OUI), may be purchased and registered with the IEEE [3, 18], which gives the organization control of and responsibility for all addresses with a particular three-byte prefix. The manufacturer is then free to assign the remaining low-order three bytes (2^{24} distinct addresses) any value they wish when initializing devices, subject to the condition that they do not use the same MAC address twice.

An implication of the IEEE registration system is that it is trivial to look up the manufacturer of a device given its MAC address. Using, again, the example of a wireless eavesdropper, this means that anyone listening to 802.11 traffic can determine the manufacturer of nearby devices. To combat this, the IEEE also provides the ability to purchase a "private" OUI which does not include the company's name in the register. However, this additional privacy feature is not currently used by any major manufacturers that we are aware of.

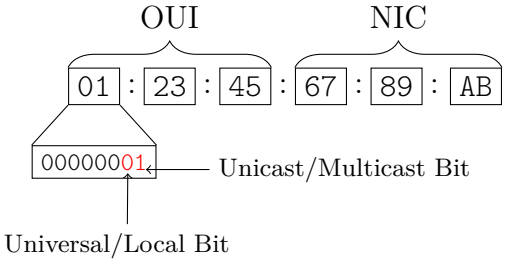

Fig. 1. 48-bit MAC Address Structure

In addition to the public, globally unique, and manufacturer assigned MAC address, modern devices frequently use *locally assigned* addresses [8] which are distinguished by a Universal/Local bit in the most significant byte. Locally assigned addresses are not guaranteed to be unique, and generally are not used in a persistent manner. Locally assigned addresses are used in a variety of contexts, including multi-Service Set IDentifier

(SSID) configured access points (APs), mobile device-tethered hotspots, and peer-to-peer (P2P) services. A visual depiction of the MAC address byte structure is illustrated in Figure 1.

Most importantly for this paper, locally assigned addresses may also be used to create randomized MAC addresses as an additional measure of privacy. Similar to an OUI, a three-byte Company Identifier (CID) prefix can be purchased from the IEEE, with the agreement that assignment from this address space will not be used for globally unique applications [3]. As such, a CID always has the local bit set, and is predisposed for use within MAC address randomization schemas. One such example, the Google owned `DA:A1:19` CID [18], is prominent within our dataset.

With the advent of randomized, locally assigned MAC addresses that change over time, tracking a wireless device is no longer trivial. For this reason, we frequently observe 802.11 probe requests using locally assigned addresses when the device is in a disassociated state (not associated with an AP). When a mobile device attempts to connect to an AP, however, it reverts to using its globally unique MAC address. As such, tracking smartphones becomes trivial while they are operating in an associated state.

Since mobile devices are usually only associated while the user is relatively stationary (otherwise they would be out of range of the AP), tracking them in this state is less of a privacy vulnerability than having the ability to track devices in an unassociated state, which usually occurs when the user is moving from one location to another. Additionally, there are several good reasons to use a global address in an associated state, such as to support MAC address filtering on the network. Therefore we concentrate, in this paper, on evaluating randomization methods and tracking of unassociated devices.

2.2 Mobile OS MAC Randomization

A particularly sensitive privacy issue arises from the manner in which wireless devices identify access points within close proximity. Traditionally, devices perform *active scanning* where they broadcast probe request frames asking nearby APs to identify themselves and respond with 802.11 parameter information required for connection setup. These probe request frames require a source MAC address, but if an 802.11 device uses its globally unique MAC address then it is effectively broadcasting its identity at all times to any wireless receiver that is nearby. Wireless device users can then easily be tracked across temporal and spatial boundaries as their devices are transmitting with their unique identity.

To combat this privacy concern, both Android and Apple iOS operating systems allow for devices in a disassociated state to use random, locally assigned MAC addresses when performing active scans. Since the MAC address is now random, users gain a measure of anonymity up until they associate with an AP.

The particular software hooks used for randomization vary between operating systems. See Appendix A for a discussion of the OS mechanisms and configuration files that support MAC randomization.

3 Related Work

Vanhoef et al. [28] present several techniques for tracking devices regardless of privacy countermeasures such as MAC address randomization. These attacks rely on devices' support for Wi-Fi Protected Setup (WPS), a protocol that allows unauthenticated devices to negotiate a secure connection with access points. Unfortunately, in order to facilitate this process, extra WPS fields are added in a device's probe requests that contain useful information for device tracking. Among these is the manufacturer and model of the device, but also a unique identifier called the Universally Unique IDentifier-Enrollee (UUID-E) which is used to establish WPS connections. The flaw that Vanhoef et al. [28] discovered is that the UUID-E is derived from a device's global MAC address, and by using pre-computed hash tables an attacker can simply lookup the UUID-E from the table and retrieve the global MAC address [20, 28]. We refer to this technique as *UUID-E reversal*. Since the UUID-E does not change, the implication is that even if the MAC address is randomized, an attacker can still recover the original, global address by performing this reversal technique on the UUID-E.

While the revelation of the flaw was significant, several holes in the analysis were observed due to the dataset on which the work was evaluated. The attack was applied against an anonymized dataset from 2013 [9]. This dataset did not include randomized MAC address implementations as they did not exist in 2013. Additionally, due to the fact that the data was anonymized, and ground truth was not available, a validation of the reversal technique was not provided. The authors state that the address could not be confirmed to

be the WiFi MAC address, rather it may represent the Bluetooth MAC address of the device. Because of this, the reader is left with little understanding on the scope of practical use of these attacks. Namely, is the attack truly viable against devices performing randomization?

The first contribution of this paper is a better evaluation of the attacks presented by Vanhoef et al. [28]. Using more recent real-world data, we verify that this technique is plausible for defeating randomization for a small set of devices. However, we also show that an improvement on their technique can achieve a higher success rate, up 99.9% effectiveness against vulnerable devices. We are also able to confirm that the retrieved MAC address is in fact the 802.11 WiFi identifier and not the Bluetooth address using additional techniques. More importantly, we provide a real-world assessment for the scope of the attack, revealing that only a small portion of Android devices are actually vulnerable.

Vanhoef et al. [28] present an additional technique: fingerprinting of the probe request 802.11 Information Elements (IEs). IEs are optional, variable length fields which appear in WiFi management frames and are generally used to implement extensions and special features on top of the standard WiFi protocol. Importantly, there are enough of these extensions and manufacturer specific functions that the various combinations which are supported on a particular device may be unique to that device, causing the IEs to form a fingerprint which can be used to identify traffic coming from that device.

However, we find one significant flaw in the evaluation of these fingerprints: locally assigned MAC addresses were ignored by the authors. Nearly all randomization schemes utilize locally assigned MAC addresses to perform randomization. As such, previous research failed to identify problems observed when tracking randomized MAC addresses. A simple example of this is the signature of a device's probe request, which we observed changing during randomization and even when not randomizing. Only by observing these behaviors can we truly implement effective derandomization techniques and present honest reflections on the limitations of the attack methods.

Also presented in [28] is a revival of the Karma attack using a top-n popular SSID honeypot approach. As noted above, MAC randomization stops once a device becomes associated with an AP. Karma attacks are active attacks where a rogue AP is configured with an identical name (SSID) to one that the device is set up to automatically connect to [12]. In effect, this forces the devices into an authenticated state where it reveals its global MAC address and bypass randomization. We validate this attack by finding that the increased prevalence of seamless WiFi-offloading from cellular networks means that many devices in the wild are vulnerable.

A set of related work [12, 14, 21] explores the efficacy of observing and evaluating the timing of probe request frame transmissions in order to fingerprint wireless devices. Franklin et al. [14] specifically focus on identification of the device driver for a device's Network Interface Card (NIC). In an effort to extend this work to tracking devices performing MAC address randomization, Matte et al. [21] explore inter-arrival times of successive probe requests frames. They present a novel timing-based attack that requires no derived layer-2 or above attributes. The dataset however, was transformed and derived from non-randomized MAC addresses. We posit that the lack of true randomized MAC address datasets limits the accuracy and ability to understand the current state of practice in MAC address randomization, the inherent flaws, and the feasibility of derived attacks.

We find that previous methods, relying on higher layer traffic attributes [26] prove unsuitable for defeating MAC address randomization. Mobile device MAC address randomization implementations occur while a device operates in an unassociated state and therefore do not transmit data frames, effectively eliminating these attack methods. Furthermore security and privacy countermeasures implemented by the OS have largely rendered SSID profiling attacks [13] moot. These attacks rely on deriving the unique SSIDs a device *seeks* when transmitting directed probes during active scanning. The practice of transmitting directed probe requests has largely been eliminated by the OS. Where directed probes are still used in practice we find the feasibility of SSID-based attacks to be practically limited to the aforementioned Karma attack.

4 Methodology

Our initial goal is to identify which mobile devices are using randomization, in order to narrow down further investigation into their exact methods for doing so. Since this is not a capability that is advertised in a specification, we resort to broad capture and analysis of WiFi traffic in order to determine which device models are doing randomization.

Over the course of approximately two years, we captured unencrypted 802.11 device traffic using inexpensive commodity hardware and open-source software. We primarily use an LG Nexus 5 Android phone run-

ning Kismet *PcapCapture* paired with an AWUS036H 802.11b/g Alfa card. We hop between the 2.4GHz channels 1, 6, and 11 to maximize coverage. We additionally employ several Raspberry Pi devices running Kismet with individual wireless cards each dedicated to channels 1, 6, and 11. Our corpus spans January 2015 to December 2016 and encompasses approximately 9,000 individual packet captures. The collection contains over 600 gigabytes (GBs) of 802.11 traffic, consisting of over 2.8 million unique devices.

It is important to note that, since devices only randomize when they are unassociated, the only traffic we are interested in is 802.11 management frames and unencrypted multicast Domain Name System (mDNS) packets. Therefore we did not capture actual intentional user traffic from the device, i.e. web browsing, email, etc., but only automatic, non-personal traffic sent by the device.

4.1 Ethical Considerations

Our collection methodology is entirely passive. At no time did we attempt to decrypt any data, or perform active actions to stimulate or alter normal network behavior while outside of our lab environment. However, given the nature of our data collection, we consulted with our Institutional Review Board (IRB).

The primary concerns of the IRB centered on: i) the information collected; and ii) whether the experiment collects data "about whom" or "about what." Because we limit our analysis to 802.11 management frames and unencrypted mDNS packets, we do not observe Personally Identifiable Information (PII). Although we observe IP addresses, our experiment does not use these layer-3 addresses. Even with an IP address, we have no reasonable way to map the address to an individual. Further, humans are incidental to our experimentation as our interest is in the randomization of wireless device layer-2 MAC addresses, or "what." Again, we have no way to map MAC addresses to individuals.

Finally, in consideration of beneficence and respect for persons, our work presents no expectation of harm, while the concomitant opportunity for network measurement and security provides a societal benefit. Our experiment was therefore determined to not be human subject research.

4.2 Identifying Randomization

We know devices implement MAC randomization in different ways. In order to quantify the vulnerabilities of employed randomization policies, we first attempt to categorize devices into different *bins*, with identical behavior, so that we can investigate characteristics of these individual techniques and seek to identify flaws in their implementation. For instance, as we will see, all iOS devices fall into the same bin, in that they handle randomization in a similar way. Android devices, on the other hand, differ significantly from iOS, and also vary greatly from manufacturer to manufacturer.

Our first step is to identify whether a device is performing randomization. This starts with extracting all source MAC addresses derived from probe request frames in our corpus. If the local bit of the MAC address is set, we store the address as a locally assigned MAC address in our database. Since randomized addresses cannot be unique, we assume at this point that any device using randomization will set the local bit in its MAC address and therefore all randomization candidates will be in this data set. For each address we then parse the advertised WPS `manufacturer`, `model_name`, `model_number`, and `uuid_e` values. Additionally, we build signatures derived from a mapping of the advertised 802.11 IE vendor fields using techniques from related work in device-model classification [16, 28]. Each MAC address, associated WPS values (when applicable), and the device IE signature are stored in our database.

Our device signatures are created using custom built Wireshark dissectors to parse the 802.11 vendor IE fields and values. Our modifications to standard wireshark files (*packet-ieee80211.c* and *packet-ieee80211.h*) allow us to efficiently create the individual device signatures as we process the packet captures, eliminating any need for post-processing scripts. Furthermore, this allows us to use a signature as a display filter while capturing. We will later use the device signatures for both passive and active derandomization techniques.

Our corpus contained a total of ∼66 million individual probe requests. We have a dataset of 2.6 million unique source MAC addresses after removing duplicates. In Table 1 we observe that 1.4 million (∼53%) of the 2.6 million distinct MAC addresses had locally assigned MAC addresses. Recall that locally assigned addresses are not only used for randomization. Therefore, after partitioning the corpus, we separate the 4,371 locally assigned MAC addresses that are used for services such as P2P and WiFi-Extenders from those used

Table 1. Corpus Statistics

Category	# MACs
Corpus	2,604,901
Globally Unique	1,204,148
Locally Assigned	1,400,753

Table 2. Locally Assigned Bins

Category	# MACs
Locally Assigned	1,400,753
Randomized	1,388,656
Service	4,371
Malformed	6,895
Unknown	831

Table 3. Randomization Bins

Category	# MACs
Randomized	1,388,656
Android: `DA:A1:19` (WPS)	8,761
Android: `DA:A1:19`	43,924
Android: `92:68:C3` (WPS)	8,961
iOS	1,326,951
Windows 10 / Linux	59

as randomized addresses for privacy purposes. Doing so required us to manually inspect the frame attributes and look for identifying characteristics.

One prevalent P2P service that makes use of locally assigned addresses is WiFi-Direct. Fortunately, WiFi-Direct operations contain a WiFi-Direct IE (`0x506f9a,10`). Specifically, the following attributes are are observed with all WiFi-Direct traffic: i) WiFi-Direct IE is present, ii) the observed OUI is simply the original OUI with the local bit set, and iii) the SSID value, if observed, is set with a prefix of `DIRECT-`. Furthermore, manual inspection of the packet capture reveals that these devices use a single locally assigned MAC address for all observed probe request frames. As these devices are not conducting randomization we remove them from our dataset.

Similarly, Nintendo devices operating in a P2P mode are observed utilizing a locally assigned address. Associated frames use a modified Nintendo OUI, one with the local bit set. Additionally, all Nintendo P2P probe requests contain a unique Vendor Specific IE, `0x00:1F:32`, allowing for an efficient identification and removal from our dataset.

The remainder of our service-based locally assigned addresses were attributed to WiFi extenders forwarding client probe requests. These were also identified as modifying their original OUI by setting the local bit. Commonly observed OUIs, such as Cisco, D-Link, and Belkin indicated a likely association to infrastructure devices. We confirmed our assumptions through manual packet analysis, which showed: i) the MAC address never changes, ii) each unique device probes for only one SSID, and iii) devices with WPS attributes clearly indicate wireless extender models.

The 6,895 frames labeled as *malformed* had improper frame control bits set or incorrect Frame Check Sequences (FCSs). The remaining 831 *unknown* MAC addresses contained insufficient context for us to accurately categorize. We suspect that a portion of the unknown devices are Windows 10 or Linux laptops.

Table 2 illustrates that 99.12% of all locally assigned mac addresses are randomized addresses, representing ∼53% of our total corpus. While this may seem like it indicates a large rate of adoption for MAC randomization, these addresses do not directly correlate to the number of unique devices in our dataset. While globally unique addresses have a 1-to-1 relationship with individual devices, a device performing randomization has a 1-to-many relationship. It is plausible that a device conducting randomization may have tens of thousands of addresses over a collection period. To illustrate this behavior we observe in Section 5.1.2 where 8,761 `DA:A1:19` MAC addresses resolve to 2,341 unique devices, a roughly 4:1 reduction. Similarly, we observe 8,961 `92:68:C3` MAC addresses reduce to 849 distinct devices, a more than 10:1 reduction. Therefore, while ∼53% of our corpus is made up of randomized addresses, we posit that a significantly smaller representation of devices conduct randomization.

Our goal, to identify and evaluate potential flaws in currently fielded randomization policies, requires that we must first answer non-trivial questions about our real-world dataset. How many devices were actually performing randomization? Which manufacturers and models have implemented randomization in practice and why? What operating systems are prevalent? Which randomization policies are actually used?

As discussed above, we must first identify distinct *bins* of randomization within the data. Table 3 highlights the results of this analysis. We completed this analysis by evaluating the following; i) the MAC address prefix (OUI, CID, random), ii) WPS attributes, iii) 802.11 IE derived device signatures, and iv) mDNS fingerprinting techniques [20]. Lastly, we confirm our analysis using devices procured by our team and evaluated in a controlled Radio Frequency (RF) environment. We provide detailed analysis of our methods, results, and answers to our stated questions in §5.

5 Analysis

5.1 Android Randomization

After removing all of the service-based locally assigned MAC addresses described in §4.2, we aim to separate the remaining ∼1.388 million addresses into distinct bins. First we perform a simple query of our database where we identify the most common three byte prefixes. We expect that the prefixes with the highest occurrences will be the CID owned by the representative devices. Our findings were surprising: first, the Google owned CID `DA:A1:19`, was by far the most commonly observed prefix (52,595), while the second most common prefix `92:68:C3`, observed 8,691 times was not an IEEE allocated CID, but rather a Motorola owned OUI with the local bit set [18].

The remaining 177k observed three-byte prefixes, each with total occurrences ranging from a low of two to a high of seven, show no indication of being a defined prefix or CID. While we expected to see the Google owned CID, we also expected to see additional CIDs configured by manufacturers to override the default Google CID.

5.1.1 `92:68:C3`

Investigating the `92:68:C3` prefix in more detail, we see that devices using this prefix always transmit granular WPS details. This is helpful as it lets us easily determine the device model (see §3). First, the Motorola Nexus 6 is the only device using this prefix. Using the WPS derived UUID-E as a unique identifier, we see that there were 849 individual Motorola Nexus 6 devices in our dataset. Second, in order to retrieve the global MAC address we use the UUID-E reversal technique previously mentioned [20, 28]. We find that the actual prefix of the device's MAC address is not the expected `90:68:C3` OUI. Rather, we observe a set of different Motorola owned OUIs. In combination with with the *config.xml* file (see Appendix A) retrieved from publicly available repositories we identify that the prefix `92:68:C3` was purposefully set by Motorola to replace the Google owned CID.

Searching open source Android code repositories revealed no additional *config.xml* defined prefixes other than the Google and Motorola ones. This matches what we observe in our real-world dataset.

Table 4. `DA:A1:19` Manufacturer Breakdown

Manufacturer	Total Devices	Model Diversity
Huawei	1,708	11
Sony	277	23
BlackBerry	234	4
HTC	108	2
Google	13	2
LG	1	1

5.1.2 `DA:A1:19`

The analysis of the Google CID `DA:A1:19` proved more complex, having serious implications to prior work in derandomization attacks. Unlike the Motorola prefix, not all devices using the Google CID transmit WPS attributes. This had multiple effects on our analysis. First, we were unable to easily identify the manufacturer and model information when no WPS information was present. Lacking a UUID-E, we were unable to precisely identify total device counts. More importantly, we were unable to retrieve the global MAC address via the reversal technique. Surprisingly, only ∼19% of observed MAC addresses with the Google CID contain UUID-E values. Since the reversal technique of Vanhoef et al. [28] require a UUID-E, this emphasizes the fact that previous evaluations are insufficient. A large majority of Android phones are not vulnerable to UUID-E reversal, despite how valuable the technique initially seems.

We evaluated the 8,761 addresses that have WPS values before attempting to breakdown the 43,924 `DA:A1:19` MAC addresses with no WPS information. We observed a diverse, yet limited spread of manufacturers and models, depicted in Table 4. Huawei was the most prevalent manufacturer observed, primarily attributed to the (Google) Nexus 6P (1660 unique devices). Various versions of the Huawei Mate and Huawei P9 were also commonly observed. Sony was well-represented with 277 unique devices across 23 variations of Xperia models. There were several surprising observations in this list, namely that Samsung was absent despite having the largest market share for Android manufacturers [23]. Blackberry, HTC, and LG were also poorly represented. The Blackberry device models were actually four derivations of the Blackberry Priv, accounting for 277 unique devices observed. HTC was largely represented by the HTC Nexus 9 from the Google Nexus line, which explains the likely use of randomization. The HTC One M10 was the remaining HTC device and was only observed once. The only observed LG device was the LG G4 model. We provide a full device breakdown in Appendix C.

In all, devices having randomized MAC addresses with a Google CID and containing WPS attributes amount to a total of 2,341 unique devices. Taking into account the 849 unique Motorola Nexus 6 devices, only 3,188 devices spanning 44 unique models are susceptible to the UUID-E reversal attack. Effectively, ∼99.98% of the locally assigned MAC addresses in our corpus are not vulnerable to the UUID-E attack. Furthermore, our corpus contains approximately 1.2 million client devices with globally unique MAC addresses and over 600 manufacturers and 3,200 distinct models using WPS data fields. This begs the question, are a large number of Android devices not conducting randomization? Do we expect the 43,924 randomized addresses using the Google CID that did not not transmit WPS information to make up all remaining Android devices?

We attempt to answer these questions by evaluating the 43,924 DA:A1:19 MAC addresses where no WPS derived data is available. The process proceeds as follows:

1. Divide the entire bin into segments, based on the device's signature described in §4.2, resulting in 67 distinct device signatures, with a starting hypothesis that each signature represents a distinct model of phone.
2. For each signature, parse every packet capture file where that device signature and the CID DA:A1:19 were observed.
3. Apply to our parsing filter our custom Tshark device signature and limit to probe request frames.

The output of the algorithm is the source MAC address, sequence number, SSID, and device signature.

Left with 2,858 output files, each mapping a device signature with distinct packet capture, we systematically retrieve the global MAC addresses for the randomized devices. We will describe in detail the methods for derandomization for this portion of the dataset in §6. After we obtain the global MAC address for the set of randomized MAC addresses within each bin, we attempt to identify the device model using a variety of techniques. It is trivial to identify the manufacturer as the OUI provides sufficient resolution. However, in order to conjecture as to the device model we borrow from the work of [20] in which we obtain model granularity from MAC address decomposition. Next, we look for any case where a device using a global MAC address as the source of a probe request matches the desired signature and also transmitted a mDNS packet at some point. For this subset we simply retrieve the model information from the mDNS packet [20]. This leaves us with guesses as to what devices randomize MAC addresses using the DA:A1:19 CID and transmit no granular WPS-derived model data. We posit that our set of 67 signature bins can be condensed into groups of similar signatures based on our derived model correlations.

Table 5. DA:A1:19 no WPS

Category	Confirmed	% of no WPS
Bin 1		57.7%
LG Nexus 5X	√	
Google Pixel	√	
Bin 2		18.5%
LG G5	√	
LG G4	√	
Bin 3		2.0%
OnePlus 3	√	
Xiaomi Mi Note Pro	√	
Bin 4		.2%
Huawei	√	
Sony	√	
Bin 5		2.6%
Cat S60	√	
Bin 6		12.2%
Composite	√	
Bin 7		6.8%
Unknown		

In order to better evaluate our assumptions, and now that we have a smaller, manageable set of possible devices, we procure devices for lab testing. We test each device using an RF enclosed chamber to ensure we limit our collection to only our individual test phones. We leave each device in the chamber for approximately five minutes, collecting only the probe requests.

We evaluate the collection results by comparing to our derived signatures and ask the following: do we observe MAC address randomization? If so, does the device signature match expectations when using a global address? Similarly, does the device signature match expectations when using a randomized address? Our findings are presented in Table 5.

Bin 1 is represented by the Google devices LG Nexus 5X and Google Pixel. This bin encompasses 57.7% of the 43,924 MAC addresses observed using the Google CID without WPS data. It is prudent to mention that we cannot claim that is an exhaustive list of devices implementing randomization using this set of signatures.

Next, we evaluate bin 2, representing 18.5% of the category's total. We observe only LG devices, specifically we posit that LG G series devices make up this subset. We confirm that both the LG G4 and G5 devices match the signatures and behavior of this bin. We

surmise that additional G series devices are represented, however we have no validation at this time. Worth mentioning is that the LG G4 and Pixel identified in the previous DA:A1:19 with WPS section were only observed because a WPS action was triggered. By default, WPS data is not transmitted by the devices in our *no-wps* category. We confirm this analysis in our lab environment, observing WPS data fields only when the user triggers a WPS event.

In bin 3, a smaller bin (2%), the OnePlus 3, and the Xiamoi Mi Note Pro are representative of the identified signatures.

Bin 4, the smallest of our bins with less than one percent of our dataset, consisted of Huawei and Sony devices. These are devices seen using WPS, but in some frames do not include the WPS data fields.

The Cat S60 smartphone was the only device identified in bin 5. As in other bins, we make no claim that no other devices share this signature.

Bin 6 represents a combination of the aforementioned devices observed in the various bins. This is caused by a device, that on occasion rotate between a standard device signature and a stripped down version with limited 802.11 IE fields. An example of this signature behavior is described in §6.4 and depicted in Figure 2. As such, this bin is represented by the previously mentioned devices.

We fail to identify anything with any sense of confidence within bin 7.

5.1.3 Motorola

After an exhaustive look at the randomization schemes employed by Android we still lack any evidence of MAC address randomization by Samsung or Motorola devices (other than the Google based Motorola Nexus 6). We attempt to find any evidence of non-standard randomization employed by these models by looking at probe requests with globally assigned MAC addresses. In a similar manner to how we identified the most common prefixes for locally assigned addresses, we attempt to identify OUIs with unusually high occurrences within individual packet captures. Our premise is that this will indicate the use of an OUI as a prefix for a set of randomized MAC addresses.

We first ruled out all P2P service related addresses as previously described, leaving a single manufacturer of interest - Motorola. We identified multiple occurrences of various Motorola OUIs with an abnormally high percentage of the unique addresses in a packet capture. After inspecting forty captures with this anomaly we confirmed that a subset of Motorola devices perform randomization using neither a CID nor an OUI with the local bit set. These devices used one of several Motorola owned OUIs, using the global MAC address occasionally, and a new randomized MAC address when transmitting probe requests.

This is an especially strange result because it shows that Motorola is using randomized global addresses. This violates the core expectation that no two devices will use the same global MAC address. In particular, it is possible for one of these devices to temporarily use the true, global MAC address of another device as one of its random addresses.

We identified two distinct signatures consistently observed within this Motorola dataset. Using the aforementioned mDNS techniques to guess a device model we posit that one signature belongs to the Moto G4 model while the second corresponds to a Moto E2. We acquired Moto G4 and E2 smartphones and confirmed our hypothesis. Additionally, we observed that a Moto Z2 Play device model shares the same randomization behavior and signature as the Moto G4.

5.1.4 Samsung

It is interesting to note that we never observed Samsung devices performing MAC address randomization, despite being the leading manufacturer of Android smartphones [2]. Samsung uses their own 802.11 chipsets, so it is possible that chipset compatibility issues prevent implementing randomized MACs addresses.

Samsung devices ($200k^+$) represent ∼17% of client devices and ∼23% of non-Apple devices in our data set, contributing substantially to the low adoption rate that we see. Our observations closely match Samsung's 2016 third quarter market share of ∼21% [2]. In our lab setting we confirm Samsung's lack of randomization when tested against a wide range of Samsung models and OS versions.

5.2 iOS Randomization

After completing the randomization analysis of Android devices, we still have over 1.3 million MAC addresses not attributed to any randomization scheme. Next we turn to the analysis of iOS randomization.

Upon the release of iOS 8.0, Apple introduced MAC address randomization, continuing with minor but valu-

able updates to the policy across subsequent iOS releases. We were faced with an immediate dilemma, how do we identify iOS associated probe requests? Apple iOS devices do not transmit WPS fields to indicate any sort of model information, and we had no knowledge of any Apple owned CID. In order to identify any prefix pattern we once again utilized our RF-clean environment to test Apple device behavior. Our goal was to create as many randomized MAC addresses as possible from a device and look for a pattern in the resulting prefixes. To force a new randomized MAC address we simply enable and disable WiFi mode repeatedly.

Our initial thought was that Apple would use an OUI or CID like other manufacturers and simply randomize the least significant 24 bits of the MAC address. However, we quickly found that the MAC addresses randomly generated by iOS devices do not share any common prefix. In fact, they appear to be completely random, including the 24 OUI bits, except for the local bit which is always set to 1 and the multicast bit which is set to 0. To lend credence to this new hypothesis we sampled 47,255 random MAC addresses from an iOS device and ran standard statistical tests to determine if they were uniformly distributed (see Appendix B). These tests confirmed that, with the exception of the local and unicast bit, iOS most likely implements true randomization across the entire MAC address. This is interesting given the fact that the IEEE licenses CID prefixes for a price, meaning that Apple is freely making use of address space that other companies have paid for.

Based on these findings, we are faced with identifying a randomization scheme where randomness is applied across 2^{46} bits of the byte structure. We can not simply assume that if the prefix does not match an offset of an allocated OUI that it is an iOS device. This is due to the aforementioned clobbering of other manufacturers OUI space. Our next step was to leverage the use of mDNS once again. We take the union of global MAC addresses derived from probe requests that are also seen as source addresses for iOS related mDNS packets. This results in a set of probe requests that we can confirm are Apple iOS devices. We then extract all of the signatures for these devices. We suspected that this retrieved only a portion of the relevant iOS signatures. Next we collected signatures from all of our Apple iOS lab test devices using our RF enclosure. Finally, we identify signatures of all remaining locally assigned MAC addresses in which we have no assigned categorization. We then seek to find any probe requests with global source address that have matching signatures. If the OUI of the global addresses resolves to an Apple OUI we consider that a valid signature. This is slightly different than our mDNS test as we cannot attribute the signature to a specific set of iOS device models. We test our entire iOS signature set and ensure that no non-iOS global MAC addresses are ever observed with these signatures.

In June 2016, midway through our research, iOS 10 was released. Inexplicably the addition of an Apple vendor specific IE was added to all transmitted probe requests. This made identification of iOS 10 Apple devices trivial regardless of the use of MAC address randomization. We believe the difficulty of identifying MAC address randomization to be one of the best countermeasures to defeating randomization. The data field associated with this IE never changes across devices, providing no ability to discern distinct devices. However, it trivially confirms that the frame originated from an Apple iOS device where prior methods of identification were laborious.

Using our combined set of all Apple iOS signatures, we identify ∼1.3 million distinct randomized MAC addresses, by far the most populous (94.7%) of our randomization categories.

5.3 Windows 10 and Linux Randomization

To conclude our categorization of randomization schemes, we look to identify the probe requests from devices using Windows 10 and Linux MAC address randomization implementations. Our first test compares the signatures obtained from laboratory laptops to the signatures of our locally assigned dataset. We find 59 matches to our laptop signatures, indicating possible Windows 10 or Linux randomization. Next, we parse collection files using the locally assigned MAC addresses from the probe request frames of these devices. Our hypothesis, if we find matching locally assigned MAC addresses in authentication, association, or data frames, that the randomizations scheme is likely Windows 10 or Linux. This assumption is due to the fact that the randomization policies use the same locally assigned address for network establishment and higher layer data frames. To that end, we find that 14 of the 59 devices assessed to be Windows/Linux computers use a locally assigned MAC address when associated to a network.

6 MAC Randomization Flaws

Now that we have a baseline understanding of the randomization implementations used by modern mobile OSs we are able to assess for vulnerabilities.

6.1 Adoption Rate

The most glaring observation, while not necessarily a flaw, per se, is that the overwhelming majority of Android devices are not implementing the available randomization capabilities built into the Android OS. We expect that this may be partly due to 802.11 chipset and firmware incompatibilities. However, some non-randomizing devices share the same chipsets as those implementing randomization, so it is not entirely clear why they are not utilizing randomization. Clearly, no effort by an attacker is required to target these devices.

6.2 Global Probe Request

We next explore the flaws of the observed MAC address randomization schemes. One such flaw, the inexplicable transmission of the global MAC address in tandem with the use of randomized MAC addresses. We observe this flaw across the gamut of Android devices. The single device in which we do not observe this was the Cat S60 smartphone. In no instance did the Cat S60 transmit a global MAC address probe request, except immediately prior to an association attempt. Exploiting this flaw it was trivial to link the global and randomized MAC addresses using our device signatures and sequence number analysis. Between probe requests, the sequence numbers increase predictably so an entire series of random addresses can be linked with a global address by just following the chain of sequence numbers. While using sequence numbers has been discussed before in prior work [28], the fact that the global MAC address is utilized while in a supposedly randomized scan state has not. This strange behavior is a substantial flaw, and effectively negates any privacy benefits obtained from randomization.

In our lab environment we observed that in addition to periodic global MAC addressed probe requests, we were able to force the transmission of additional such probes for all Android devices. First, anytime the user simply turned on the screen, a set of global probe requests were transmitted. An active user, in effect, renders randomization moot, eliminating the privacy countermeasure all together. Second, if the phone received a call, regardless of whether the user answers the call, global probe requests are transmitted. While it may not always be practical for an attacker to actively stimulate the phone in this manner, it is unfortunate and disconcerting that device activity unrelated to WiFi causes unexpected consequences for user privacy.

6.3 UUID-E Reversal

Vanhoef et al. [28] introduce the UUID-E reversal attack against Android devices. Devices transmitting probe request frames with WPS enriched data fields, specifically, the UUID-E are vulnerable to a reversal attack where the global MAC address can be retrieved using the WPS UUID-E value. The flaw is caused by the construction of the UUID-E, where the MAC address is used as an input variable along with a non-random hard-coded seed value. This implementation design flaw allows for the computation of pre-computed hash tables, whereby retrieving the global MAC address requires only a simple search of the hash tables. This revelation, while both groundbreaking and disconcerting, still leaves the reader to guess as to the plausibility of the attack against randomized devices. We find several issues with their approach, specifically in respect to derandomization analysis: i) randomization was not employed in 2013, when the data used in their evaluation was gathered ii) anonymized data eliminates accuracy checks, and iii) removing locally assigned MAC addresses effectively eliminates the ability to evaluate the attack against devices performing randomization.

Accordingly, we use our corpus of `DA:A1:19` and `92:68:C3` datasets to evaluate the effectiveness and viability of the UUID-E attack. Our foremost observation is that only 29% of random MAC addresses from Android devices include WPS attributes. Effectively 71% of this Android dataset is completely immune to the UUID-E reversal attack. This is in addition to the fact that iOS devices are wholly immune to the attack, as they do not use WPS. We refer back to Table 4 the limited number of Android models performing randomization and transmitting the necessary WPS UUID-E attribute.

We then retrieve the global MAC address from the probe requests of these devices that used both random and global MAC addresses, exploiting the previously discussed flaw. We use this set of 1,417 *ground truth* MAC addresses to test the effectiveness of the UUID-E reversal attack. First we pre-compute the required hash tables. To build hash tables for the entire IEEE space would be non-trivial, requiring significant disk space and

$Sig_G = 0,1,50,3,45,221(0x50f2,8),htcap:012c,htagg:03,htmcs:000000ff$
$Sig_R = 0,1,50$

Fig. 2. Device Signature (Motorola Moto E2)

processing time. While an exhaustive compilation of the address space is certainly possible, we use the knowledge gained from decomposing the randomization schemes to efficiently construct our tables. We build the hash tables using only the OUIs owned by manufacturers we have observed to implement randomization. The resulting hash table is a manageable 2.5TBs, where using presorting techniques, we can retrieve an UUID-E's global MAC address in < 1 second.

We retrieve a global MAC address for 3,187 of the 3,188 UUID-Es. In previous work it was left inconclusive whether the retrieved MAC addresses were in fact the global 802.11 MAC address or instead the Bluetooth MAC address. The UUID-E derived from the HTC One M10 device, was the example UUID-E listed in the *wpa_supplicant.conf* file. With exception of the HTC Nexus 9, all HTC phones in our dataset (regardless of randomization) used this non unique UUID-E.

Comparing the 1,417 *ground truth* addresses to those retrieved from the UUID-E attack we achieve a 100% success rate. Indicating that the retrieved addresses are in fact the global 802.11 MAC addresses, completing the missing link from the evaluation of Vanhoef et al. [28].

6.4 Device Signature

To aide in derandomization we employ fingerprinting techniques, using signatures derived from the 802.11 IEs borrowed from previous work [16, 28]. We used this technique first to aide in the identification of the randomization schemes employed by Android and iOS devices.

This technique allows us to remove all extraneous probe request traffic, providing us a "cleaner" dataset in which to employ sequence number analysis. We modify the Wireshark files *packet-ieee80211.c* and *packet-ieee80211.h*, creating a new dissector filter, *device.signature*. We are able to filter previous collection files as well as conduct filtering on live collection. While our contribution to the Wireshark distribution is novel, the fingerprinting technique is not, as we borrowed from related work. However, prior work tested against datasets not performing randomization which fails to provide accurate context. We test the signature technique against our real world corpus, revealing flaws in previous signature based attacks.

Regardless of the Android implementation, a device transmits probe request frames which have varying signatures (based on IEs, see §3). Devices often use two or more signatures while using a global MAC address, so simply using the signature is insufficient. Additionally, the same holds for randomized addresses, in which we observe multiple signatures. In both cases, the second signature, has minimal 802.11 IEs. Due to the fact that nearly all devices periodically use this signature, it creates significant complexity to any signature based derandomization attack. Finally, as Figure 2 illustrates, we observe that most Android devices use different signatures when randomizing compared to when using a global MAC address. As such, previously described signature-based tracking methods fail to correlate the addresses. Using our decomposition of Android randomization schemes, and the derived knowledge of how distinct bins of devices behave, we properly pair the signatures of probe requests using global and randomized MAC addresses. Only by combining these signatures are we able to accurately and efficiently retrieve the global MAC address.

We observe no such change in signatures of iOS devices within a collection timeframe. While an iOS device may not use alternate signatures, they do not send globally addressed probe requests. Therefore, at this juncture, we have not identified a method of resolving the global MAC address.

6.5 Association/Authentication Frames

We observe that Android and iOS devices use sequential sequence numbers across management frame types. Using only passive analysis we can follow a devices transition from randomized probe requests to an authentication or association frame by following the sequence numbers. This is particularly useful as all authentication and association frames from iOS and Android devices use the global MAC address. Using the techniques described in [16] we create a set of signatures for the association frames of iOS devices, specifically to aide in confirmation that the device observed in the probe request is also the same device type as the association frame. This

method relies on the targeted device attempting to establish a network connection with a nearby AP. As this is fairly user-activity dependent, we reinvestigate the plausibility of the Karma attack against current randomization schemes.

6.6 Karma Attack

The current versions of iOS and Android randomization policies have eliminated the vast majority of cases where a *directed probe* is used. A directed probe is a probe request containing a specified SSID that the device wishes to establish a connection (a previously known or configured SSID), as opposed to a broadcast probe which solicits a response from all APs in range. Today, the predominant use of broadcast probes has directly effected the ability for a Karma-based attack to succeed. Karma-based attacks work by simulating an access point that a device prefers to connect to. A variety of implications such as man-in-the-middle attacks are common follow-on consequences, however we are only interested in retrieving the global MAC address and therefore require only a single authentication frame to be transmitted by a targeted device. To this end Vanhoef et al. [28] also investigate Karma attacks, implemented via a predefined top-n SSID attack, achieving a 17.4% success rate, albeit not specifically related to devices performing randomization.

Unlike previous work, we observe devices while in a randomized state in order to identify specific behaviors that directly counteract randomization privacy goals. Specifically, do we observe traits that allow for a targeted Karma attack? It is well known that hidden networks require directed probes, so while this is a vulnerability to randomization, it is fairly uncommon, and a decision in which a user chooses to implement. Similarly, previous connections to ad hoc networks, saved to the devices network list, cause both Android and iOS devices to send directed probes. As with hidden networks, this uncommon condition requires action from the user, however when observed, the Karma attack is viable.

Finally, we observe a more disconcerting trend: devices configured for seamless cellular to WiFi data-offloading, such as Hotspot 2.0, EAP-SIM and EAP-AKA force the use of directed probes and are inherently vulnerable to Karma-based attacks [6]. The expanding growth of such handover polices reveals a significant vulnerability to randomization countermeasures. Further exasperating the problem, these devices are preconfigured with these settings, requiring no user interaction. We confirmed these settings by inspecting the *wpa_supplicant.conf* file of a Motorola Nexus 6 and Nexus 5X. Removing the networks from the configuration file requires deletion by a rare user with both command line savvy and awareness of this issue.

We test for the presence of these network configurations in our corpus by evaluating all randomized addresses using WPS fields. We are able to accurately evaluate unique devices using the UUID-E value as the unique identifier. We filter for any instance where the device sends a directed probe, retrieving the SSID value for each. Sorting by most common occurrence the top three most common SSIDs were `BELL_WIFI`, `5099251212`, and `attwifibn`. The SSIDs `BELL_WIFI` and `5099251212` are used by the mobile carrier Bell Canada for seamless WiFi offloading. Interestingly, the `attwifibn` SSID is related to free WiFi hotspots provided by the Barnes and Noble bookstore. Only ∼5% of the 3,188 devices transmitted a directed probe. However, of those that did, 17% were caused by the preconfigured mobile provider settings.

6.7 Control Frame Attack

We now evaluate active attack methods for identifying a device by its global MAC address while in a randomized state. Our premise: can we force a device performing MAC address randomization to respond to frames targeting the global MAC address? This would allow for easy tracking of devices, even when they are randomizing, because an active attacker could elicit a specific response from them at any time if they are within wireless range.

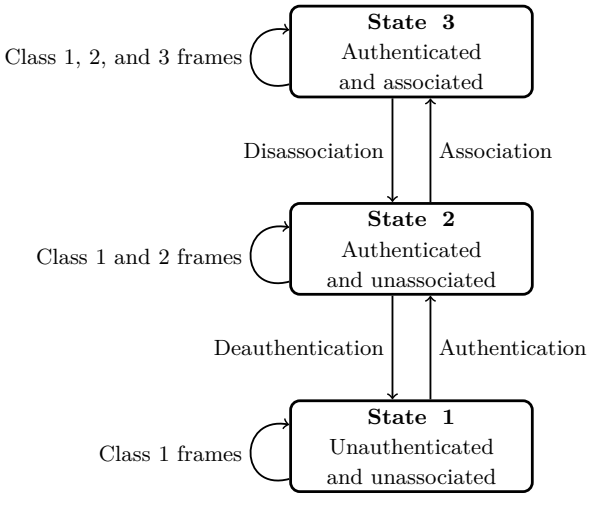

Fig. 3. 802.11 State Diagram

Table 6. Class 1 Frames [15]

Control	Management	Data
RTS	Probe Request	Frame w/DS bits false
CTS	Probe Response	
Ack	Beacon	
CF-End	Authentication	
CF-End+CF-Ack	Deauthentication	
	ATIM	

Figure 3 depicts the 802.11 state diagram illustrating the various states of association for 802.11 devices [15]. We are particularly interested in the frame types that can be sent or received while in an unauthenticated and unassociated state (State 1). The frame types (Class 1 frames) allowed while in State 1 are depicted in Table 6.

In our lab environment, we use packet crafting tools (SCAPY, libtins) to transmit customized packets for each frame type, targeting the global MAC of the device.

The source MAC address of the frame is a uniquely crafted MAC address. It is not the actual MAC address of our transmitter. This ensures that we can accurately track any responses to our crafted message, removing any possible control frames that happen to be sent to the actual transmitter address. Of the twelve Class 1 frame types used for the attack, we successfully elicited a response from only the Request-to-Send (RTS) frame.

Request to Send and Clear to Send (RTS/Clear-to-Send (CTS)) transmissions are available in the IEEE 802.11 specification as part of a Carrier Sense Multiple Access with Collision Avoidance scheme. When a node desires to send data an RTS may be sent to alert other nodes on the channel that a transmission is about to begin and the period of time during which they should not transmit on that channel so as to avoid collisions. If there are no conflicting uses of the channel, the target node will respond with a CTS to acknowledge the request and give the transmitting node permission to solely communicate on the medium.

As for previous location and tracking attacks, some researchers have used RTS/CTS messages to perform Time of Arrival computations [17] while others have extended these techniques to perform Time Difference of Arrival calculations from timestamps in exchanged frames [11]. These older methods perform localization on Access Points from client devices. Musa and Eriksson [24] present a basic RTS injection attack in order to elicit a response from a client device. The novelty in our method is that we are sending RTS frames to 802.11 client devices while in a randomized state. We can extract a CTS response message which we derive the true global MAC address of that device, effectively using RTS/CTS exchanges to perform derandomization attacks.

The result of sending a RTS frame to the global MAC address of a device performing randomization was that the target device responded with a CTS frame. A CTS frame, having no source MAC address, is confirmed as a response to our attack based on the fact that it was sent to the original, crafted source MAC address [24]. A full device listing utilized for the control frame attack is available in Appendix D.

If the global MAC address is known, that device can be easily tracked just as if randomization were never enabled. This might cause one to wonder why vendors would go to such lengths to include MAC address randomization in a device only to allow that same device to divulge the protected information through an administrative protocol. We assert that this phenomenon is beyond the control of individual vendors. The fact is that this behavior occurs across the board on every device we have physically tested as shown in Appendix D. This leads us to believe that RTS/CTS responses are not a function of the OS, but of the underlying IEEE 802.11 chipset. Manufacturers have configured their chipset hardware with default RTS/CTS operation which may not even be accessible to configure at the OS level. If we are correct, this derandomization issue can not be fixed with a simple patch or OS update. Susceptible mobile devices will be unmasked by this method for the lifetime of the device. Additionally, due to the hardware level nature of this phenomenon, there will be a significant delay in the market until mobile devices resistant to this attack are produced, assuming manufacturers recognize this as a flaw and subsequently design a process truly capable of delivering MAC address privacy.

There are multiple scenarios in which a motivated attacker could use this method to violate the privacy of an unsuspecting user. If the global MAC address for a user is ever known, it can then be added to a database for future tracking. This global MAC address can be divulged using the techniques discussed in this paper, but it can also be observed any time the user is legitimately using that global MAC address, such as when connected to an AP at home or work. This single leakage of the true identifier will allow an attacker to send an RTS frame containing that global MAC address in the future to which that host will respond with a correct CTS when it is in range. Conceivably, an adversary with a sufficiently large database and advanced trans-

mission capabilities could render randomization protections moot. Additional testing of the control frame attack, while the target device had WiFi or Airplane-modes, enabled or disabled respectively, revealed further concerns. Namely, Android devices performing location-service enabled functions wake the 802.11 radio. Our RTS attack was thusly able to trigger a CTS response from the target, circumventing even extreme privacy countermeasures. Apple iOS devices failed to elicit CTS responses when the device was in Airplane mode, WiFi was disabled, or the WiFi radio was in a sleep state. Lastly, we add improvements, using our Wireshark signature filters, to eliminate the constant barrage of transmitted RTS frames. Our collection algorithm is preloaded with the target of interest's device signature, where upon observing the signature in the target area we launch the preconfigured MAC address. We test this against our diverse test phones with 100% success.

6.7.1 Bluetooth Correlation

We offer an additional method to derive the global WiFi MAC address for later use in a RTS attack. Wright and Cache [29] claim that Apple iPhone devices, beginning with the iPhone 3G, utilize a one-off scheme for the allocation of the Bluetooth and WiFi MAC addresses, where the MAC address is actually equal to the Bluetooth address, plus or minus one. Using a novel algorithm to calculate the WiFi and Bluetooth MAC address from iOS devices operating in hotspot mode, we provide evidence countering this claim.

We identified that Apple iOS devices, operating in hotspot mode, send beacon management frames containing an Apple vendor specific IE. This *Type 6* field closely resembles the source MAC address of the device. As Wireshark does not process this field correctly we built custom dissectors to create display filters for the Apple vendor tag IE and associated data fields. We first test on 29 Apple iOS lab devices, placing each in hotspot mode and collecting the beacon frames. We retrieve the true Bluetooth and WiFi MAC addresses from the device settings menu of the phone. We then parse the beacon frames, outputting the source MAC address and six byte Type 6 IE.

We observe that the Type 6 field exactly represents the Bluetooth MAC address. The source MAC address of the Beacon frame has the local bit set. However, the first byte of the source MAC address is not a simple offset of the global MAC address as seen in most P2P operations. To resolve the actual global MAC address we find that replacing the first byte of the source MAC address with the first byte of the Type 6 (Bluetooth Derived) MAC address, we obtain the correct WiFi MAC address of the device. This permutation is successfully tested for all 29 test devices across the gamut of model and iOS versions.

Interestingly, six of the 29 test devices did not show a one-off MAC address allocation. As such, we seek to identify the accuracy of the previous claim that iOS devices use this one-off scheme by evaluating across our entire corpus.

A total of 3,576 devices were identified in our dataset containing the Type 6 field of which ∼95.4% utilized a one-off addressing scheme. Interestingly, ∼88.2% of those devices had a Bluetooth address that was one-higher than the WiFi MAC address. Indicating that even when the offset is used it is not uniformly implemented. We are unsure as to why ∼4.6% of iOS devices do not use the one-off policy. Regardless, in all cases the OUI of the two interfaces are the same. Using the mDNS model correlation analysis we observed no indication that offset scheme is correlated with the device model.

7 Conclusions

We provide a detailed breakdown of the randomization polices implemented, the associated device models, and the identification methods thereof. This granularly detailed decomposition allowed for fine-tuned improvements to prior attempts at MAC address derandomization as well as providing novel additions.

Our analysis illustrates that MAC address randomization policies are neither universally implemented nor effective at eliminating privacy concerns. Table 7 depicts the diversity of presented attacks, across the spectra of randomization schemes and OSs, highlighted by the RTS control frame attack targeting a widespread low-level chipset vulnerability. Active attacks are, by definition, harder to execute without being noticed. Each attack has its own set of necessary conditions, as described above, but we have organized Table 7 roughly in order of severity from left being the most severe to right being the least severe.

We conclude that Android devices are susceptible to the spectrum of passive and active derandomization techniques. Samsung devices do not conduct randomization at all, failing to provide a modicum of identifier obfuscation. Conversely, iOS devices, while broken

Table 7. Derandomization Technique Results
Attacks can be carried out by a passive adversary unless otherwise noted

Randomization Bin	Global MAC Address Probe Request	UUID-E Reversal	Auth/Assoc Frames	Hotspot 2.0 - Karma Attack (*Active*)	RTS Attack (*Active*)
`DA:A1:19` with WPS	√	√	√	√	√
`DA:A1:19` w/o WPS	√	×	√	√	√
`92:68:C3` with WPS	√	√	√	√	√
Motorola (No local bit)	√	×	√	√	√
Apple iOS	×	×	√	√	√

for some edge cases, require specific network interaction and/or active attacks for defeating randomization implementations.

To be truly effective, randomization should be universally adopted. A continued lack of adoption, allowing for simpler identification, effectively reduces the problem set for an attacker. The more devices performing randomization within a test set, the harder it will be to diffuse each device's associated traffic. This is particularly true if we can continue to bin the various schemes, further reducing the problem set.

We propose the following best practices for MAC address randomization. Firstly, mandate a universal randomization policy to be used across the spectra of 802.11 client devices. We have illustrated that when vendors implement unique MAC address randomization schemes it becomes easier to identify and track those devices. A universal policy must include at minimum, rules for randomized MAC address byte structure, 802.11 IE usage, and sequence number behavior.

To reiterate, these best practices can only be truly effective when enforced across the spectrum of devices. Granular examples of such policy rules:

- Randomize across the entire address space, providing 2^{46} bits of randomization.
- Use a random address for every probe request frame.
- Remove sequence numbers from probe requests or set the sequence number to a fixed value for all probe request frames.
- If sequence numbers are used, reset sequence number when transmitting authentication and association frames.
- Never send probe requests using a global MAC address.
- Enforce a policy requiring a minimal standard set of vendor IEs. Move any lost functionality to the authentication/association process, or upon network establishment utilize discovery protocols.
- Specifically, the use of WPS attributes should be removed except when performing P2P operations. Prohibit unique vendor tags such as those introduced by Apple iOS 10.
- Eliminate the use of directed probe requests for cellular offloading AP discovery.
- Mandate that chipset firmware remove behavior where RTS frames received while in State 1 elicit a CTS response.

Acknowledgments

We thank Rob Beverly, Adam Aviv, and Dan Roche for early feedback. Views and conclusions are those of the authors and should not be interpreted as representing the official policies or position of the U.S. government. The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

References

[1] Linux WPA supplicant (IEEE 802.1x, WPA, WPA2, RSN, IEEE 802.11i). https://w1.fi/wpa_supplicant/.

[2] IDC: Smartphone vendor market share. http://www.idc.com/promo/smartphone-market-share/vendor.

[3] Guidelines for Use Organizationally Unique Identifier (OUI) and Company ID (CID). https://standards.ieee.org/develop/regauth/tut/eui.pdf.

[4] WPA supplicant change log. https://w1.fi/cgit/hostap/plain/wpa_supplicant/ChangeLog.

[5] China Deputizes Smart Phones to Spy on Beijing Residents' Real-Time Location. https://www.eff.org/deeplinks/2011/03/china-deputizes-smart-phones-spy-beijing-residents, Oct 2011.

[6] WiFiGate - How Mobile Carriers Expose Us to Wi-Fi Attacks. https://www.skycure.com/blog/wifigate-how-mobile-carriers-expose-us-to-wi-fi-attacks/, Apr 2014.

[7] Danger Close: Fancy Bear Tracking of Ukrainian Field Artillery Units. https://www.crowdstrike.com/blog/danger-

close-fancy-bear-tracking-ukrainian-field-artillery-units/, Jan 2017.

[8] D. E. 3rd and J. Abley. IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters. RFC 7042 (Best Current Practice), Oct. 2013.

[9] M. V. Barbera, A. Epasto, A. Mei, S. Kosta, V. C. Perta, and J. Stefa. CRAWDAD dataset sapienza/probe-requests. http://crawdad.org/sapienza/probe-requests/20130910, Sept. 2013.

[10] J. Bard. Unpacking the Dirtbox: Confronting Cell Phone Location Tracking with the Fourth Amendment. *BCL Rev.*, 57:731, 2016.

[11] Z. Cui and A. Agrawala. WiFi Localization Based on IEEE 802.11 RTS/CTS Mechanism. In *Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems*, pages 199–208. ICST, 2015.

[12] M. Cunche. I know your mac address: targeted tracking of individual using wi-fi. *Journal of Computer Virology and Hacking Techniques*, 2014.

[13] M. Cunche, M. A. Kaafar, and R. Boreli. Linking wireless devices using information contained in Wi-Fi probe requests. In *Pervasive and Mobile Computing, vol. 11*, pages 56–69, 2014.

[14] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting.

[15] M. Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly, Beijing, Farnham, 2005. ISBN 0-596-10052-3.

[16] D. Gentry and A. Pennarun. Passive Taxonomy of Wifi Clients using MLME Frame Contents. *CoRR*, abs/1608.01725, 2016.

[17] C. Hoene and J. Willmann. Four-way TOA and software-based trilateration of IEEE 802.11 devices. In *2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–6, Sept 2008.

[18] IEEE. OUI Public Listing. http://standards.ieee.org/develop/regauth/oui/oui.txt.

[19] D. Kerr. Russian police spy on people's mobile data to catch thieves. https://www.cnet.com/news/russian-police-spy-on-peoples-mobile-data-to-catch-thieves/, Jul 2013.

[20] J. Martin, E. Rye, and R. Beverly. Decomposition of MAC Address Structure for Granular Device Inference. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 78–88. ACM, 2016.

[21] C. Matte, M. Cunche, F. Rousseau, and M. Vanhoef. Defeating MAC Address Randomization Through Timing Attacks. In *Proceedings of the 9th ACM Conference on Security; Privacy in Wireless and Mobile Networks*, WiSec '16, pages 15–20. ACM, 2016.

[22] C. Mims. If You Have a Smart Phone, Anyone Can Now Track Your Every Move. https://www.technologyreview.com/s/427687/if-you-have-a-smart-phone-anyone-can-now-track-your-every-move/, Oct 2012.

[23] T. Mitchell. Smartphone ownership rates skyrocket in many emerging economies, but digital divide remains. http://www.pewglobal.org/2016/02/22/smartphone-ownership-rates-skyrocket-in-many-emerging-economies-but-digital-divide-remains/, Feb 2016.

[24] A. Musa and J. Eriksson. Tracking Unmodified Smartphones Using Wi-Fi Monitors. In *Proceedings of the 10th ACM conference on embedded network sensor systems*, pages 281–294. ACM, 2012.

[25] B. L. Owsley. Spies in the Skies: Dirtboxes and Airplane Electronic Surveillance. *Mich. L. Rev. First Impressions*, 113: 75–75, 2015.

[26] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 User Fingerprinting. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 99–110, 2007.

[27] M. Sarwar and T. R. Soomro. Impact of Smartphone's on Society. *European journal of scientific research*, 98(2):216–226, 2013.

[28] M. Vanhoef, C. Matte, M. Cunche, L. Cardoso, and F. Piessens. Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms. In *ACM AsiaCCS*, 2016.

[29] J. Wright and J. Cache. *Hacking Exposed Wireless: Wireless Security Secrets & Solutions*. McGraw-Hill Education Group, 3rd edition, 2015. ISBN 0071827633, 9780071827638.

A OS Randomization Configuration

A.1 Android

In October 2014 the *wpa_suppplicant.conf* file, used by Android, Linux, Windows, and OS X client stations [1] for configuration of 802.11 networking, was updated to add experimental support for MAC address randomization in network scans. Full implementation support was added in March 2015 [4]. Listing 1 depicts the added support for MAC address randomization. It is worth noting that the configuration file provides two policies for using a non-globally unique address while in an associated state. If the variable `mac_addr` is set to *1* the device will use a randomized MAC address for each unique network the device connects to. If `mac_addr` is set to *2* the device will randomize the lower three bytes of the MAC address prefixed with the original OUI where the local bit has been set to 1.

The *wpa_supplicant.conf* file also addresses the randomization policies available for disassociated devices conducting active scanning. In this case, the variable `preassoc_mac_addr` can be set similarly to the previously described address policies.

Listing 1. wpa_supplicant.conf

```
# MAC address policy default
# 0 = use permanent MAC address
# 1 = use random MAC address for each ESS connection
# 2 = like 1, but maintain OUI (with local admin bit
    set)
#
# By default, permanent MAC address is used unless
    policy is changed by
# the per-network mac_addr parameter. Global mac_addr=1
    can be used to
# change this default behavior.
```

```
#mac_addr=0

# Lifetime of random MAC address in seconds (default:
    60)
#rand_addr_lifetime=60

# MAC address policy for pre-association operations
    (scanning, ANQP)
# 0 = use permanent MAC address
# 1 = use random MAC address
# 2 = like 1, but maintain OUI (with local admin bit
    set)
#preassoc_mac_addr=0
```

Android introduced MAC address randomization for probe requests with Android 6.0 (Marshmallow) and in an incremental patch to 5.0 (Lollipop). With the release of Marshmallow, the *WifiStateMachine.java* and *WifiNative.java* files were modified to implement MAC address randomization for active scanning. When the *SupplicantStartedState* function is called upon enabling WiFi, a call to the newly added *setRandomMacOui* function sets the first three bytes of the MAC address to the default Google CID (`DA:A1:19`). If the `config_wifi_random_mac_oui` variable has been redefined in the *config.xml* file, that prefix will be used in place of the default Google CID. The XML configuration file allows an Android smartphone manufacturer to override the default Google CID with a prefix to be used as the substitute for the OUI. Finally, the prefix is passed to another function, *setScanningMacOui* located in the *WifiNative.java* file which calls a corresponding function at a lower, native level. If the device chipset is compatible to support randomization then the prefix will be used during active scans.

We extracted the *wpa_supplicant.conf*, *WifiStateMachine.java*, and *WifiNative.java* files from Android devices that do and do not perform MAC address randomization. We found that the *wpa_supplicant* file was never utilized to implement randomization, as attempts to modify the randomization settings of the file had no affect on any device. The Java files also had the supporting functions for randomization included, regardless if the device used them. Interestingly, with logging enabled, the devices that did not conduct randomization sent output to the logs indicating that the random MAC had been set, where devices seen randomizing did not.

A.2 iOS

In late 2014, Apple introduced MAC address randomization with the release of iOS 8.0. Apple iOS randomization settings are not device-model customizable, unlike Android, which allows each model to modify settings such as the CID. As of the current iOS 10.x version, Apple devices only use the locally assigned MAC address while in a disassociated state. Since iOS is not open source, we cannot determine the exact method or configuration options that Apple uses on their devices to support randomization. Instead, we are left to determine device behavior from a "black box" perspective by observing communication from different devices and iOS versions in §5.

B iOS Randomization Tests

To determine if iOS is using random prefixes, or if there is just a pattern that we have not been able to see, we used several standard statistical tests to compare our observations with an ideal, random distribution. First, we calculated the number of *collisions* we observed, where the same prefix appeared more than once. If they are truly random we would expect to see a moderate number of collisions, which is easy to quantify. We would also expect to see a certain, far fewer, number of *triple collisions* where one prefix appears three times. These numbers can be calculated as follows:

$$E[\#\text{ of collisions}] = \frac{\binom{n}{2}}{m}$$

$$E[\#\text{ of triple collisions}] = \frac{\binom{n}{3}}{m^2}$$

$$\text{where } n = \#\text{ of addresses observed}$$

$$m = \#\text{ of possible prefixes } (2^{22})$$

Comparing our empirical results with the statistical expectations, we get:

$$\text{For :}$$

$$\text{Collisions : expected} = 266, \text{ observed} = 262$$

$$\text{Triple collisions : expected} = 1, \text{ observed} = 3$$

Additionally, we decomposed the bytes of subsequent MAC addresses into a bit stream and ran the tests specified in the FIPS 140-1 standard published by NIST to test random number generators. We obtained the following results:

- Monobit test: 9939
- Poker test: 13.56
- Runs test length 1: 2515
- Runs test length 2: 1342
- Runs test length 3: 581
- Runs test length 4: 281
- Runs test length 5: 166
- Longest run test: 12

All tests passed within the allowable ranges. These tests indicate to us that the MAC addresses are distributed uniformly.

C Google CID Device Breakdown

Table 8. DA:A1:19 with WPS Model Breakdown

Manufacturer	Model	Distinct Devices
Huawei	Nexus 6P	1660
BlackBerry	STV100-3	133
HTC	Nexus 9	107
BlackBerry	STV100-1	71
Sony	E5823	61
Sony	E6653	59
Sony	SO-01H	29
Sony	E6853	23
Blackberry	STV100-4	20
Huawei	NXT-L29	17
Sony	SO-02H	17
Google	Pixel C	12
Sony	SO-03H	11
Sony	SOV32	11
Huawei	NXT-AL10	11
BlackBerry	STV100-2	10
Sony	SO-03G	9
Sony	SOV31	8
Sony	E6883	8
Sony	E5803	8
Sony	E6553	7
Huawei	NXT-L09	6
Sony	E6683	6
Huawei	EVA-L09	5
Sony	F5121	5
Sony	E6533	4
Huawei	EVA-AL00	3
Huawei	KNT-AL20	2
Huawei	EVA-AL10	2
Sony	SGP712	2
Sony	SGP771	2
Sony	E6603	1
Sony	E6633	1
Sony	SO-05G	1
LGE	LG-H811	1
Sony	E6833	1
Huawei	VIE-AL10	1
Huawei	EVA-DL00	1
Sony	402SO	1
Google	Pixel XL	1
Sony	501SO	1
Huawei	EVA-L19	1
Sony	F5321	1
HTC	HTC 2PS650	1

D RTS Control Frame Attack - Device Diversity

Table 9. RTS Control Frame Attack - Device Diversity

Model	OS Version	Success
iPhone 6s	10.1.1	√
iPhone 6s	9.3.5	√
iPhone 6s Plus	9.3.5	√
iPhone 5s	10.1	√
iPhone 5s	9.3.5	√
iPhone 5	9.3.5	√
iPad Air	9.3.5	√
Google Pixel XL	7.1	√
LGE Nexus 5X	7.0	√
LGE G5	6.0.1	√
LGE G4	6.0.1	√
Motorola Nexus 6	6.0.1	√
Moto E2	5.1.1	√
Moto Z Play	6.0.1	√
OnePlus 3	6.0.1	√
Xiaomi Mi Note Pro	5.1.1	√