Hacker's Toolbox: Detecting Software-Based 802.11 Evil Twin Access Points

Fabian Lanze*, Andriy Panchenko*, Ignacio Ponce-Alcaide†, Thomas Engel*
*University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
Email: {firstname.lastname}@uni.lu
†University of Málaga
Email: iponce@uma.es

***Abstract*—The usage of public Wi-Fi hotspots has become a common routine in our everyday life. They are ubiquitous and offer fast and budget-friendly connectivity for various client devices. However, they are exposed to a severe security threat: since 802.11 identifiers (SSID, BSSID) can be easily faked, an attacker can setup an evil twin, i.e., an access point (AP) that users are unable to distinguish from a legitimate one. Once a user connects to the evil twin, he inadvertently creates a playground for various attacks such as collection of sensitive data (e.g., credit card information, passwords) or man-in-the-middle attacks even on encrypted traffic. It is particularly alarming that this security flaw has led to the development of several tools that are freely available, easy to use and allow mounting the attack from commodity client devices such as laptops, smartphones or tablets without attracting attention. In this paper we provide a detailed overview of tools that have been developed (or can be misused) to set up evil twin APs. We inspect them thoroughly in order to identify characteristics that allow them to be distinguished from legitimate hardware-based access points. Our analysis has discovered three methods for detecting software-based APs. These exploit accuracy flaws due to emulation of hardware behavior or peculiarities of the client Wi-Fi hardware they operate on. Our evaluation with 60 hardware APs and a variety of tools on different platforms reveals enormous potential for reliable detection. Furthermore, our methods can be performed on typical client hardware within a short period of time without even connecting to a potentially untrustworthy access point.**

I. Introduction

Public Wi-Fi networks have become indispensable in our daily life. They provide fast Internet access at little cost (often even free of charge) in public places such as hotels, airports or cafés. For many devices that cannot access mobile 3G/4G networks (e.g., laptops, tablets), Wi-Fi offers the only possibility for connecting to the Internet. Moreover, using such networks is essential whenever mobile networks are too slow, overpriced or not accessible due to roaming. Therefore, connecting to public Wi-Fi networks has become a usual routine for many users – despite potential security risks. As a matter of fact, the 802.11 standard does not provide strong identifiers for Wi-Fi access points. The only attributes that users or client applications can use to verify an AP's authenticity are its SSID (the network name) and the BSSID (the MAC address of the AP). These identifiers can be trivially spoofed by an attacker, enabling him to set up a *faked AP*, impersonating the legitimate AP's identity and therefore being unrecognizable by clients. This threat is commonly referred to as the *evil twin attack*. Once the user has fallen into the trap by inadvertently connecting to a faked AP, the attacker can mount various attacks. Besides harvesting and analyzing data, the establishment as intermediary enables the attacker to act as a man in the middle and, hence, to attack encrypted connections. This appears even more menacing considering the fact that users in general tend to accept unsigned or wrongly signed certificates [20]. For a categorized review of the threat model we refer the reader to [13].

Several security mechanisms for 802.11 networks exist, but they fail to provide thorough protection against evil twin attacks in public Wi-Fi networks: when **Wi-Fi Protected Access (WPA)** is applied, the communication is encrypted with a pre-shared key (PSK). Such a mechanism could only mitigate evil twin attacks if the PSK were solely distributed to legitimate clients but concealed from an attacker. However, this is generally not possible for public hotspots, where the key must be distributed to clients, e.g., printed on a receipt. In **802.1X/WPA-Enterprise** enabled networks, the authenticity is guaranteed by strong cryptographic primitives and Certification Authority certificates in particular. While such a mechanism is suitable for restricting access to corporate/enterprise environments for a limited set of users, it is not applicable to public Wi-Fi networks: it requires complex setup and maintenance, and most providers of public hotspots have no incentive to deploy such a solution. Finally, many Wi-Fi hotspots use **web-based authentication**: the first connection attempt is redirected to a website locally hosted by the operator (called a *captive portal*) that provides a disclaimer and requests login or credit card information for accounting. Nevertheless, this solution does not provide any security for the user at all: the captive portal can be simply cloned by the attacker and the information entered can be stored for further misuse. Note that this task can be fully automated using appropriate tools (see Section III).

Consequently, the research community has proposed a variety of protection mechanisms in the past decade. These works generally disregard the fact that the attacker does not necessarily have to set up the evil twin using a dedicated hardware device. Instead, several software tools exist that allow to mount the attack in multiple variants from commodity mobile devices, particularly those running Linux. As we will show, even the built-in mobile hotspot feature of current smartphones is sufficient. Such an approach is generally preferable for the attacker. On the one hand, there is no need to make additional expenditure to obtain and set up hardware if he is equipped

with a set of software tools that is capable of doing the same – or even more, since the tools are able to fully automate most aspects of the attack. On the other hand, the use of software on a mobile device minimizes to risk of being discovered and prosecuted. Therefore, we address the detection of software-based 802.11 evil twins in particular.

Our contributions are as follows:

- We provide a comprehensive overview of available tools for evil twin attacks. We analyze their capabilities and weaknesses (both experimentally and based on source code review) to disclose those that attackers most likely use in practice.
- We reveal several distinguishing marks that available tools exhibit. These are not limited to implementation errors (which could easily be avoided) but also consider inevitable characteristics, e.g., accuracy flaws caused by the emulation of hardware behavior. We propose methods that enable the detection of software-based APs, or even of the specific type of software in use.
- We experimentally validate our methods and show their detection capabilities for various platforms. To the best of our knowledge, we are the first to provide methods for the reliable detection of software-based evil twin attacks.

II. Related Work

The proposed methods to protect against evil twin attacks can be grouped into three major categories: protocol modifications, hardware fingerprinting, and non-hardware-based identification. A few approaches propose modifications to existing protocols [5], [2], [7]. They introduce concepts similar to TLS/SSL to authenticate wireless APs. To achieve protection they either require unique SSIDs that are tied to certificates or apply a trust-on-first-usage strategy similar to the SSH protocol. However, methods requiring the modification of standard protocols are undesirable as they cannot be easily deployed in existing networks.

The second category of approaches utilizes the hardware characteristics of an AP in order to create its fingerprint. Ideally, this fingerprint should be unique, even for devices of the same vendor and series. Naturally, such methods require a trusted central party to collect and manage fingerprints. Several approaches exploit an unavoidable physical phenomenon called clock skew that causes clocks based on crystal oscillators to have small yet observable deviations in speed [10], [1]. Though it was shown that a significant amount of devices share a similar clock skew, recently proposed methods [12] utilize device-intrinsic temperature dependency to significantly enrich the fingerprints' information content. Another hardware characteristic utilized to distinguish physical devices is based on radiometric signal properties of the wireless unit [4], [17]. Though these methods are extremely precise, they require dedicated specialized hardware for measurements and cannot be performed by a regular client device.

Non-hardware-based identification methods focus on the investigation of network or environmental properties of APs to detect an evil twin. Some approaches aim to detect an evil twin if it introduces an additional wireless hop on each route. These methods assume that the fake AP uses a wireless link to the legitimate AP in order to relay its Internet connection. In [8] the authors detect an extra hop on the path by analyzing the RTT to a local DNS server. Song et al. [19] consider the inter-packet arrival time to distinguish between a one-hop and a two-hop wireless channel. Mónica et al. [15] create watermarked packets to detect whether these are relayed on a different wireless channel. Note that a legitimate hotspot connected to the Internet via a wireless link would also be classified as evil twin by these methods. Another approach [11] assumes that the attacker is running several SSIDs on the same device and tries to detect this by correlating received signal strengths. In summary, the above mentioned methods only solve a part of the problem: they detect the coexistence of an evil twin in situations where the fake AP routes traffic through the legitimate AP, is connected via a wireless link to the Internet, or address situations where the attacker runs several SSIDs on the same device (although neither is necessarily a sign of a fake AP).

Another group of non-hardware-based approaches attempts to fingerprint behavioral characteristics of the AP, e.g., timings related to the authentication procedure [18] or packet inter-arrival times [16]. However, these methods either require an additional device for monitoring, or achieve relatively low detection rates of about 50%. Bratus et al. [3] utilize *malformed stimuli-response*, i.e., how devices react to manipulated or fragmented frames to create a fingerprint. However, this was a proof-of-concept without thorough evaluation. Gonzales et al. [7] focus on the group of simultaneously reachable APs instead of isolated devices. They define a context as a set of tuples containing SSID and RSSI for all APs visible from a certain location. The AP's check fails if its context has significantly changed. This approach only works when the attacker sets up the evil twin at a different location from the legitimate AP. Another method is based on the network intrusion detection system from the side of the network operator [14].

To sum up, the methods proposed so far only partially solve the problem of evil twins. Moreover, though the most typical attack scenario is to use a software-based AP, no single method (besides our initial attempt [13]) has been proposed that tries to determine whether the AP is software- or hardware-based. With our work we aim to tackle this issue.

III. Software-Based 802.11 Access Points

In this section, we present an overview of existing software tools that are either explicitly designed for or that can be misused to perform evil twin attacks in 802.11. We require considered tools to be capable of creating a fully operational 802.11 access point, i.e., a master station in an infrastructure basic service set (BSS). We refer to such a setup as *Soft-AP*. Additionally, the tools should run on available (i.e., not obsolete and therefore unsupported) hardware.

On a Linux-based operating system, a Soft-AP can be set up by combining a wireless network interface controller (WNIC) operating in *master mode* – requiring the chipset *and* the driver to support it – and **hostapd**[1], a user-space daemon

[1] http://w1.fi/hostapd/

that implements AP management and all relevant authentication methods. Hostapd supports most wireless drivers, in particular all that are built on the mac80211 framework[2]. Being comprehensively configurable, hostapd can be considered as a candidate to be misused by an attacker to set up an evil twin.

The **MadWifi**[3] drivers used to be a popular choice for WNICs with Atheros chipsets. Based on a proprietary hardware abstraction layer (HAL), they allowed control of many functions including the setup of multiple virtual APs. As one of the most advanced Wi-Fi drivers for Linux, MadWifi was also used in security research [1]. However, the MadWifi project was abandoned years ago and the drivers have been superseded by ath5k and ath9k which replace the proprietary HAL with the standard mac80211 framework and are now part of the Linux kernel. Since these do not provide advanced functionality for the setup of Soft-APs and the deprecated MadWifi drivers are not supported by modern hardware and kernels, we omit their separate evaluation.

The **aircrack-ng**[4] suite is a set of actively developed and maintained tools implementing various types of attacks in 802.11 environments. It is intended for auditing the security of wireless networks. Among other components, the suite contains a tool called **airbase-ng**, which is a Soft-AP implementation aiming at attacking clients in particular. To this end, it is able to manipulate and resend packets and to capture WPA(2) handshakes. Running on a variety of Wi-Fi chipsets and being available as Live CD, aircrack-ng can be considered to be a fully-featured toolbox that attackers could use out of the box for evil twin attacks.

Karma[5] is a set of patches which implements the ad hoc clone attack [13], where the AP creates networks on the fly that match those probed by users. It was originally written for MadWifi and later ported to hostapd. **Karmetasploit**[6] and **Katalina**[7] are implementations of Karma for airbase-ng. Besides the ad hoc clone strategy, these tools provide further components to automate sub-tasks of evil twin attacks such as configuring an appropriate DHCP server. Since all these tools are internally based on one of the two major Soft-AP implementations for Linux, i.e., airbase-ng or hostapd we omit their detailed analysis and only focus on their core – if we can detect airbase-ng and hostapd we implicitly also cover these tools.

We also disregard several tools that do not satisfy our basic requirements: **Fake AP**[8] is capable of creating multiple counterfeit APs in order to confuse network scanners. However, these APs are not operational and the tool is limited to obsolete Prism Wi-Fi chipsets. Similar limitations pertain to **rfakeap**[9], which simulates the presence of an AP at the management frame level, and **Airsnarf**[10], a proof-of-concept implementation to provide faked captive portals in order to steal users' credentials. **mdk3**[11] implements, among other attacks such as forced deauthentication, AP emulation similar to FakeAP through beacon flooding, i.e., by sending numerous beacons of short-lived virtual devices. These can cause a denial of service or crash certain network scanners, but are not intended to provide any service.

The major mobile operating systems, **Android** and **iOS**, include a built-in feature to set up a Soft-AP, called *Tethering and portable hotspot* (Android) and *Personal Hotspot* (iOS). These functions are restricted in regard to the supported number of connections and can only route traffic through the device's cellular 3G/4G connection. Although the functionality and performance are quite limited, they are still sufficient to perform evil twin attacks. Besides running appropriate interception tools on a (rooted) device, an attacker could set up practically any portable device to connect via a built-in VPN function, e.g., PPTP, to a gateway controlled by him and route the traffic of victims connected to his mobile hotspot through this tunnel. The gateway can then serve as an intermediary for man-in-the-middle attacks. This method should work on any smartphone – without the device being rooted or installing additional apps. **Windows 7 & 8** include Soft-AP functionality called *Windows Virtual Wi-Fi*. We are not aware of any specialized tools for evil twin attacks using Windows besides those to simplify the setup process of a Soft-AP. Nevertheless, the strategy described for smartphones could be easily deployed to mount the attack from a laptop running Windows.

The boundaries between Soft-APs and hardware-based APs are often blurred. As mentioned, the basic requirement for an 802.11 AP is an appropriate Wi-Fi chipset supporting master mode and suitable software for functionality, management and authentication. We define a *hardware* (or *genuine*) *AP* as an embedded device only built for this purpose. Note that the Wi-Fi chipsets used for hardware APs are not necessarily different from those embedded in client Wi-Fi adapters hosting a Soft-AP. Although specialized Wireless Systems-on-a-chip (WiSoC) exist that are explicitly designed for use in (enterprise) wireless APs, in fact, many popular chipsets such as the Atheros AR5005GS or the Broadcom BCM4318 are used for both client adapters and hardware APs[12]. Besides, it is often the case that the operating systems of hardware APs are based on Linux and use hostapd to control the functionality, e.g., OpenWRT. Still, we endeavor to explore characteristics that can distinguish between Soft-APs and genuine APs. These may be based on implementation discrepancies, inaccuracies due to emulation delays or even, as we will show in the next section, on slight differences in the firmware used for identical chipsets in different environments.

IV. Methods for Detection

In this section, we describe the particularities that Soft-APs exhibit, which we have identified either based on source code review or by experimentation. We focus our analysis on two types of management frames in 802.11: *beacon frames* and probe frames (*probe requests/responses*). Beacons are periodically sent (typically every 100 ms) by APs to announce

[2] http://wireless.kernel.org/en/developers/Documentation/mac80211
[3] http://madwifi-project.org
[4] http://www.aircrack-ng.org
[5] http://digi.ninja/karma
[6] https://dev.metasploit.com/redmine/projects/framework/wiki/Karmetasploit
[7] https://github.com/kussic/Katalina
[8] http://faculty.ccri.edu/jbernardini/JB-Website/ETEK1500/LinuxTools/FakeAPMain.htm
[9] http://rfakeap.tuxfamily.org
[10] http://airsnarf.shmoo.com
[11] http://aspj.aircrack-ng.org/#mdk3
[12] See available lists on https://wikidevi.com

TABLE I: Laptops used for our experiments

	OS	Wi-Fi chipset	Driver
Host A	Ubuntu 12.04	Atheros5002X	ath5k
Host B	Ubuntu 10.10	Broadcom BCM4312 a/b/g	b43
Host C	Backtrack RC5	Intel Pro/Wireless 3945ABG	iwl3945
Host D	Ubuntu 12.10	Broadcom BCM4313 b/g/n	b43
Host E	Ubuntu 14.04 LTS	Intel WiFi Link 5100	iwlwifi

their presence to nearby stations (passive scanning) and for timing synchronization. Such frames can be analyzed in a completely passive way, i.e., without any interaction. Probe frames, on the other hand, are explicitly sent by stations (active scanning). This enables clients to actively probe for certain APs (either known ones or APs that fulfill queried characteristics, e.g., a minimum supported rate). Limiting the evaluation to these types of management frames is desirable because they can be analyzed before initiating any connection to a potentially untrustworthy AP. An overview of the laptops used in our experiments including operating systems and Wi-Fi equipment is shown in Table I. We implemented all described methods using the Python library *scapy*[13].

A. Analyzing beacons to disclose TSF (in-)accuracy

In [13] we showed that airbase-ng is highly sensitive to accuracy flaws in *Timing Synchronization Function* (TSF) timestamps and proposed an appropriate detection method. Here, we provide a more detailed analysis of this technique and explore whether it can be transferred to other types of Soft-APs. To this end, we briefly recapitulate the background and method. An AP serves as the timing master for all associated stations. Therefore, the beacon frames it periodically emits contain a timestamp of its TSF timer in microsecond resolution and all client stations adjust their local TSF timer to this value. 802.11 specifies strict constraints for the accuracy of TSF timestamps: an AP should set the timestamp such that it equals the TSF timer at exactly that moment, when the first bit is actually transmitted [9]. We argued that whenever software needs to emulate this methodology (for which genuine APs use optimized firmware/hardware) the accuracy of generated timestamps diminishes. As a detection method we introduced the root-mean-square error (RMSE) of an ordinary least squares regression (OLSR) fitted into a trace of recorded beacons. Formally, a trace is given by $T = \{(t_{REC_1}, o_1), ..., (t_{REC_n}, o_n)\}$, where t_{REC_i} denotes the time when the ith beacon was received and $o_i = t_{TSF_i} - t_{REC_i}$ is the offset between the TSF timestamp in the ith beacon and its respective reception time. While such a trace forms an accurate linear pattern for hardware APs, we observed that traces for airbase-ng exhibit a significant number of outliers and generally a greater scattering. We were able to distinguish between airbase-ng operated on four different laptops and a set of 30 hardware APs by comparing the RMSE of a trace defined as

$$\text{RMSE}(T) = \sqrt{\frac{\sum_{i=1}^{n} (P_{LSR}^{T}(x_i) - y_i)^2}{n}}$$

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

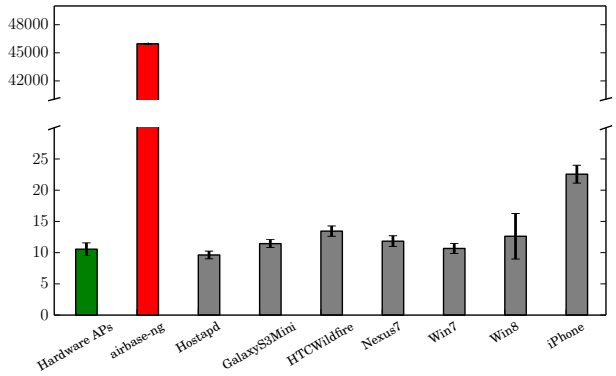


Fig. 1: Average RMSE of TSF accuracy for different Soft-AP implementations or platforms (including 95%-confidence intervals)

where $P_{LSR}^{T}(x) = \alpha x + \beta$ is the prediction function of an OLSR for that trace.

We now investigate whether this method can also detect other Soft-AP implementations. In each run, we set up one laptop to host airbase-ng or hostapd (if supported, see Section III). We then recorded 60 traces of 200 beacons each from all remaining laptops. Additionally, we set up Soft-APs on Host D with Windows 7, two additional Laptops with Windows 8, an iPhone 4, and three Android devices, a Samsung Galaxy S3 mini (Android 4.1.2), a HTC Wildfire (Android 2.3) and a Google Nexus 7 (Android 4.4) and recorded their beacon traces accordingly. Finally, we gathered 30 additional traces for hardware APs resulting in 60 different hardware APs in our dataset. The average RMSE calculated for 200 beacons is shown in Figure 1. Hardware APs show an expected low RMSE (i.e., a high accuracy) of 10.5 on average. The results for airbase-ng reflect our observations in [13]: the average RMSE is larger by several orders of magnitude and, so, reliably detectable. All other investigated Soft-APs exhibit a TSF accuracy that is comparable to hardware APs – the deviations can be attributed to measuring inaccuracies and the small number of test devices. We expect that the reason for this result is the way the Soft-APs generate beacons, particularly the value for the TSF timestamp. As mentioned above, genuine APs use an optimized combination of hard- and firmware to achieve high accuracy. airbase-ng fails to imitate this accuracy because the process of beacon generation and transmission is prone to system delays (see [13] for further details). Obviously, the other tools investigated here do not rely on emulating this process in software, but rather delegate this task to low-level drivers or the hardware. We were able to verify this in the case of hostapd[14], but could not validate it for the other tools as their source code is not publicly available.

In [13], we further observed that the RMSE magnitude varies, depending on the laptop hosting airbase-ng. We therefore performed additional experiments to narrow down the influencing factors, particularly the WNIC used. Hosts A, B and C have a PCMCIA slot allowing the WNIC to be changed and the same WNIC to be used in different hosts. We

[14]hostapd defines an interface, which is responsible for appropriate beacon generation and delivery that compatible drivers must implement.

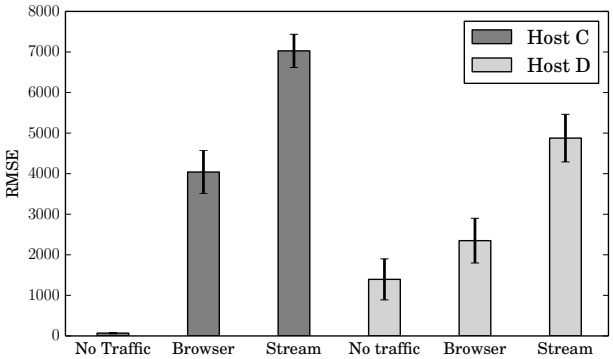


Fig. 2: RMSE of TSF accuracy for airbase-ng: two hosts under different load conditions (including 95%-confidence intervals)

experimented with two Wi-Fi cards, a 3COM 3CRWE154A72 (AR5002X) and a Netgear WG511T (AR50056s). Our results are ambiguous: we could not identify a significant correlation between the accuracy of TSF timestamps and (a) the Wi-Fi chipset or (b) the performance of the laptop (e.g., CPU speed). We even identified a combination (Host C + AR5002X) that generated TSF timestamps with an accuracy comparable to hardware APs and, therefore, produces a false negative for our detection method. However, we cannot derive a general rule from this exception since the same card did not significantly reduce the RMSE in the two other hosts. We conclude that the relation between hardware and TSF timestamp accuracy of airbase-ng is non-trivial and influenced by additional factors (e.g., driver, OS). Nevertheless, our method detects airbase-ng in the overwhelming majority of tests.

In the experiments described above, the airbase-ng instances did not have any clients connected[15]. This is not necessarily realistic. Since the attacker mounts the evil twin attack in order to lure unsuspecting users to connect to the Internet through his AP, it is likely that there are already clients connected before a new client associates. We expect any type of traffic on the fake AP to further decrease the TSF accuracy of airbase-ng (as it is mainly affected by processing delays) and, therefore, to improve the detection efficacy of our method. To verify this, we conducted an additional set of experiments. Before measuring the RMSE of a host operating airbase-ng, we associated another laptop to the Soft-AP and simulated two types of typical load patterns: first, a user surfing the web (simulated by a script that randomly selects and retrieves websites from the most popular URLs[16]); and second, a user watching a video stream (resulting in a constant bitrate of about 1500kbit/s). The resulting RMSE for two different airbase-ng hosts is shown in Figure 2. It clearly validates our assumption: traffic further decreases the TSF accuracy of airbase-ng. Note that load has no influence on hardware APs in respect of this metric: these APs set the timestamp to the TSF timer's current value *while* the beacon is actually sent [9]. Therefore, their accuracy does not depend on delays induced by system load or wireless traffic. Finally, we conclude that our proposed detection technique performs best under realistic settings: the more users are trapped by an airbase-ng evil twin, the more easily it can be detected. As shown in [13], the RMSE stabilizes after receiving 50-100 beacons. Therefore, in practice the test takes only a few seconds.

[15]Note that for measuring the RMSE of TSF timestamp accuracy not even the device performing the measurement needs to be associated.

[16]according to http://www.alexa.com/topsites

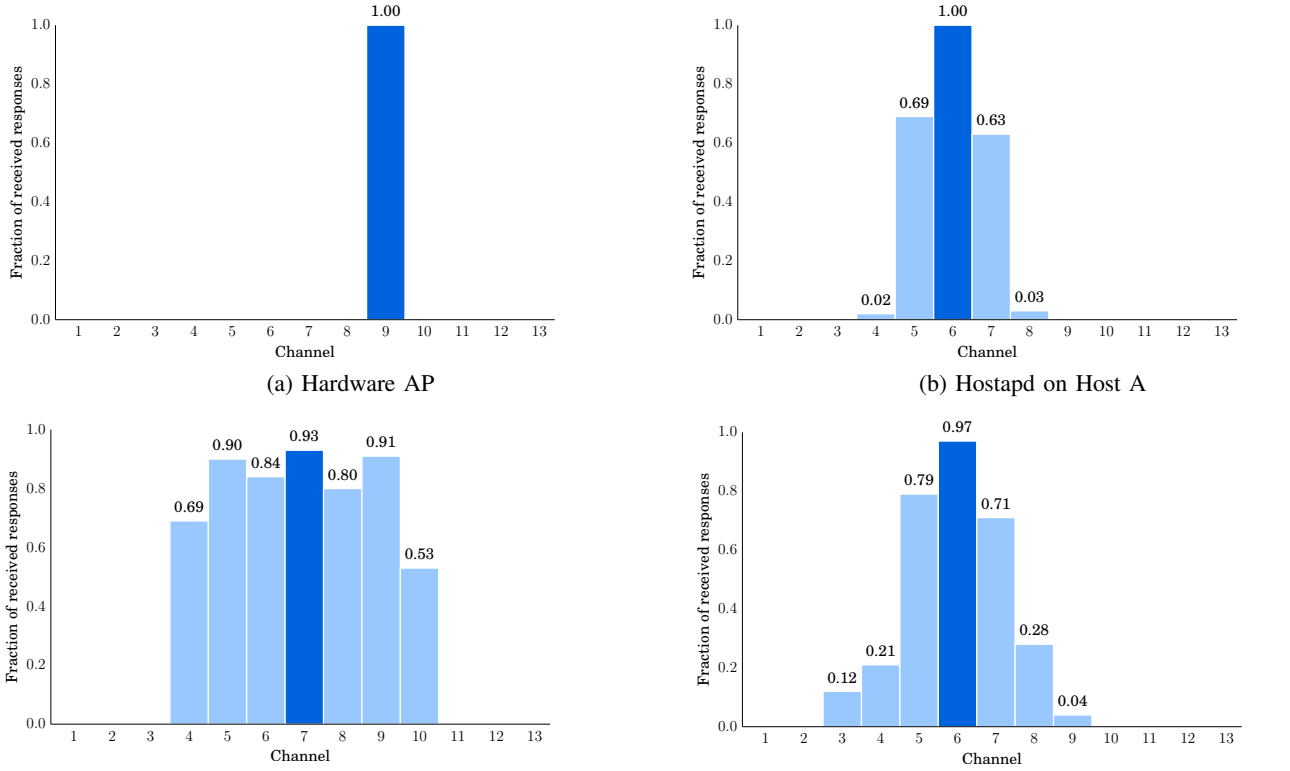
B. Probe frames

Probe requests are actively sent by a client and can either be directed, i.e., include the SSID/BSSID of a concrete AP, or broadcast by using wildcard entries. In preliminary experiments we investigated many aspects of probes to find characteristics that separate Soft-APs from genuine APs. Besides construction and compliance with specifications, we also generated probe requests, where fields were intentionally set to incorrect or empty values – a technique that is commonly referred to as *malformed stimuli-response*. During these tests, we identified two characteristics that can be exploited to detect Soft-APs.

1) Active probing on adjacent channels: Public Wi-Fi hotspots typically operate according to the 802.11g/n standard in the 2.4 GHz band as this ensures compatibility with most client devices. This band is divided into 14 channels. As the protocols require 16.25–22 MHz of channel separation but channel center frequencies are separated by only 5 Mhz, adjacent channels overlap. Therefore, it is practically unavoidable that a Wi-Fi device receives frames that were not sent on its operating channel. This is referred to as *adjacent-channel interference* and occurs in various radio environments. We observed that hardware APs and Soft-APs treat such frames differently.

We experimented by sending 100 probe requests on the operating channel as well as the corresponding adjacent channels to various hardware APs and Soft-APs. Sample results are shown in Figure 3, where the histogram bars indicate the fraction of probes that were responded to on the respective channel. We made the same observation in almost all experiments: while hardware APs always only respond on their operating channel, Soft-APs also respond on adjacent channels. The fraction of responses on these channels roughly corresponds to the fraction of frames that are expected to be received due to channel overlapping. Probe request frames may optionally contain a DSSS parameter set element containing a current channel value. In our experiments we did not see a difference according to whether we used this field or how we set it.

In general, it is not possible for a Wi-Fi receiver to determine through which exact channel a frame was received. In order to select valid frames sent through the operating channel and to filter out noise received through adjacent channels, wireless stations calculate the *adjacent channel rejection* by measuring the signal strength and comparing the synchronization of the signal modulation (for further details, we refer the reader to [9]). We presume that the firmware used for Wi-Fi chipsets in hardware APs is optimized to filter out adjacent channel interference more strictly compared to client adapters. This seems reasonable: since APs generally operate on a fixed channel, they require noise to be filtered as much as possible. Client adapters on the other hand spend a significant amount of time scanning for networks. In this scanning process, the reception of signals in adjacent channels is beneficial as it leads to faster network discovery. Although it is known that vendors use different firmware for their chipsets [6], e.g., to



(a) Hardware AP

(b) Hostapd on Host A

(c) airbase-ng on Host C

(d) Motorola Moto G hotspot mode (Android 4.3)

Fig. 3: Probe requests responded by different access points on their operating channel (dark) and adjacent channels (light)

restrict functionality for economic reasons, we could not verify our assumption since the firmware implementation is vendor-specific and generally not publicly available.

We propose to exploit this fact in the following way: assume a client device D attempts to connect to an AP A and wants to test whether it is operated by a Soft-AP. An AP's operating channel is denoted as op_A. Further, we define an estimation function $\mathrm{e}(c_1, c_2)$ for two channels $c_1 \neq c_2$ as the width of their actual overlapping frequency range divided by the channel width. In the 2.4 Ghz band for a channel width of 20 Mhz and center spacing of 5 Mhz between channels this leads to, e.g., $\mathrm{e}(6,7) = 0.75$ or $\mathrm{e}(1,3) = 0.5$. D can easily determine op_A and the set of its adjacent channels $\mathrm{AC}_A = \{c \,|\, \mathrm{e}(c, \mathrm{op}_A) > 0\}$. e serves as rough estimation of the fraction of frames that are expected to be received from an adjacent channel. D then sends N probe requests directed to A on all these channels and determines for each channel c the response rate $\mathrm{resp}_A(c)$ as the number of corresponding probe responses divided by N. Finally, D verifies that the following equation holds for a threshold θ:

$$\frac{\sum_{c \in \mathrm{AC}_A} \mathrm{resp}_A(c)/\mathrm{e}(c, \mathrm{op}_A)}{|\mathrm{AC}_A|} \leq \theta \cdot \mathrm{resp}_A(\mathrm{op}_A).$$

By testing whether the weighted mean response rate on adjacent channels is larger than a fraction of the response rate on the operating channel, a Soft-AP can be detected. It is important to consider $\mathrm{resp}_A(\mathrm{op}_A)$ to take account of the case where the target AP is distantly located or the band is overloaded, leading to a generally lower response rate. We propose to use a threshold $\theta = 0.1$. This leaves some room for hardware APs with a less restrictive filter but still reliably detects the Soft-APs in our dataset. For a delay of Δ_p between consecutive probes, the required time for the measurement is bounded by $\Delta_p \cdot N \cdot (|\mathrm{AC}_A| + 1)$. Therefore, with $\Delta_p = 100$ ms and $N = 10$ the test can be performed in a few seconds. By varying N, a trade-off can be defined between performance and precision of the test.

2) Malformed Probe Request Stimuli: In general, in our experiments, hardware APs and Soft-APs showed equal reactions to malformed probe request frames. Both implement the same specification and their compliance is necessary to guarantee adequate functionality. However, we made an interesting observation. 802.11 frames contain four 48-bit address fields which are set to different values depending on the frame type and destination. In probe requests, the Address 1 field contains the destination MAC address (either directed or broadcast) and the Address 3 field is set to the BSSID (either concrete or wildcard). Therefore, Address 3 is only relevant for members of an independent BSS (*IBSS*, i.e., an ad hoc network) or a mesh network, since in an infrastructure BSS, the BSSID *is* the MAC address of the AP's WNIC and, hence, both address entries are identical. The specification clearly states that an AP receiving a directed probe request should only respond if Address 1 and Address 3 contain its MAC address. However, we observed that hardware APs do not check the Address 3 field in probe requests, while several Soft-AP implementations including Windows and iOS do. This seems comprehensible: since APs are in general not intended to be part of an IBSS or

TABLE II: Effectiveness of our methods

	TSF	Adj. Channel	Addr3
airbase-ng	X	X	
Hostapd		X	X
Android			
Android 2.3 (Samsung Galaxy Mini)		X	X
Android 4.1.2 (Samsung Galaxy S3 Mini)		X	X
Android 4.2.2 (Samsung Galaxy S4 Mini)			
Android 4.3 (Motorola Moto G)		X	
Android 4.3 (Samsung Galaxy S3)		X	
Android 4.4 (Google Nexus 7)		X	
iOS			
iPhone 4		X	X
Windows Virtual WiFi			
Windows 7		X	X
Windows 8.1		X	X

mesh network, the value of the Address 3 field is irrelevant. Therefore, verification can be avoided to improve efficiency. Surprisingly, several Soft-APs can be detected because they obviously avoid this optimization and strictly implement the specification. This can be easily tested by a client: it only needs to send some probe requests with a manipulated Address 3 entry (e.g., a random MAC address). The outcome is binary – either the AP responds to such frames or it does not respond to any. The latter case can then be interpreted as a sign of a Soft-AP.

V. Evaluation and Discussion

Table II summarizes our results. Estimating the accuracy of TSF timestamps in beacons yields a powerful detection mechanism for airbase-ng, as it seems to be the only tool that implements this time-critical part itself, while the others delegate this functionality to drivers or the hardware. Obviously, it would be possible to adapt the way airbase-ng generates beacons to the same way as other tools, leading to undetectability by this method. However, we argue that such a modification is not desirable. The fact that airbase-ng does not depend on low-level functionality is one of its most important unique features: it can operate on almost arbitrary Wi-Fi hardware, making it particularly attractive.

Performing active probing on adjacent channels reveals striking potential for detecting Soft-APs as it does not depend on implementation issues. This method aims to detect whether the combination of firmware and chipset is optimized for hardware APs or client adapters. Our experiments with various different hardware AP models did not lead to a single false positive classification by this method. This is particularly remarkable as our dataset of hardware APs contains at least two devices running OpenWRT (hence, using hostapd internally) and, as described earlier, it is more than likely that this also applies for a significant fraction of hardware APs in general. Note that the results for hostapd in Table II only correspond to hostapd being operated on our testing laptops. As intended, hostapd is only detected as a Soft-AP when it runs on client hardware and not on a hardware AP. However, we cannot guarantee that this method is robust against false negatives (i.e., Soft-APs that are not detected), since we cannot verify which chipsets and firmware are embedded in client devices and how they concretely implement adjacent channel rejection. We identified a smartphone (Samsung Galaxy S4 mini, Android 4.2.2) that erroneously passed this test and it is to be expected that this may also apply for other devices. Still, any alarm raised by this method can be interpreted as clear warning for the user as false alarms almost never occur.

Malformed probe stimuli with manipulated Address 3 entries only detect a subset of the investigated Soft-APs. Moreover, it is not as reliable as the other methods since it exploits an implementation optimization of hardware APs that is easily reproducible by an attacker – at least when he is using open-source software. However, the required test can be done simultaneously with adjacent channel probing and the result can further corroborate the detection, especially for iOS and Windows devices.

In practice, we propose combining all three tests. The interpretation of the results depends on the situation. Of course, an access point operated by a mobile client device is not necessarily a sign for an evil twin attack. Nevertheless, when users set up their smartphone for tethering they know to which type of device they are connecting beforehand. If, on the other hand, a user in a public place intends to connect to a public Wi-Fi hotspot with an SSID *starbucks* or *mcdonalds* and our tests indicate that this hotspot is operated by a mobile device or even a tool such as airbase-ng, alarm bells should ring. As described in Section III, several hacking tools exist that enrich the capabilities of Soft-APs by implementing the ad hoc clone attack, where the AP dynamically advertises the SSID probed by clients. Fortunately, such a strategy can be easily unmasked: when a client has reasonable doubts about the authenticity of an AP, it can simply create probe requests querying for random SSIDs. Whenever these are answered, the attack is discovered.

VI. Conclusion

In this paper, we have addressed an important security threat of public Wi-Fi hotspots: the evil twin attack, where an attacker sets up a faked AP which users cannot differentiate from the legitimate one. Once lured by the malicious intermediary, it is easy to attack the client's connection and to steal sensitive data. Numerous tools can be found on the Internet that do not require special skills and can be used out of the box to mount evil twin attacks from commodity client devices. We are the first to explicitly tackle the likely scenario in which such attacks are performed using specialized software. We categorize existing tools and explore their functionality to reveal how the attack is likely to be performed in practice. airbase-ng, one of the most powerful tools in this regard exhibits significant inaccuracies in TSF timestamps. As our evaluation shows, our proposed detection method is able to reliably identify this characteristic. What is more, the exploited effect is intensified in the more realistic scenario where load on the AP is considered. Essentially all software-based APs suffer from the fact that they run on wireless network adapters that are optimized to operate as a client and not as an AP. We are able to identify their less restrictive adjacent channel rejection based filtering by actively sending probe request frames on the neighboring channels of the AP's operating frequency. Our methods can be easily implemented on typical client devices and the detection can be performed within a duration of 10–20s and without connecting to the AP. In summary, our proposed techniques can greatly improve the security for users of public Wi-Fi hotspots with minimal overhead and a vanishingly low amount of anticipated false alarms.

REFERENCES

[1] C. Arackaparambil, S. Bratus, A. Shubina, and D. Kotz, "On the reliability of wireless fingerprinting using clock skews," in *Third ACM Conference on Wireless Network Security (WiSec'10)*, 2010.

[2] K. Bauer, H. Gonzales, and D. McCoy, "Mitigating Evil Twin Attacks in 802.11," in *1st IEEE International Workshop on Information and Data Assurance (WIDA 2008) in conjunction with the 27th IEEE International Performance Computing and Communications Conference (IPCCC 2008)*, Austin, TX, USA, December 2008.

[3] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles, "Active Behavioral Fingerprinting of Wireless Devices," in *Proceedings of the First ACM Conference on Wireless Network Security (WiSec'08)*, 2008.

[4] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *14th ACM International Conference on Mobile Computing and Networking (MobiCom '08)*, 2008.

[5] T. Cross and T. Takahashi, "Secure Open Wireless Access," in *Black Hat USA 2011*.

[6] M. Gast, *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[7] H. Gonzales, K. Bauer, J. Lindqvist, D. McCoy, and D. Sicker, "Practical Defenses for Evil Twin Attacks in 802.11," in *IEEE Globecom Communications and Information Security Symposium (Globecom 2010)*, Miami, FL, December 2010.

[8] H. Han, B. Sheng, C. c. Tan, and S. Lu, "A Measurement Based Rogue AP Detection Scheme," in *28th Conference on Computer Communications (INFOCOM 2009)*, 2009.

[9] *Standard 802.11-2012: IEEE Standard for Information technology – Telecommunications and information exchange between systems, Local and metropolitan area networks – Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Computer Society, http://standards.ieee.org/findstds/standard/802.11-2012.html. [Online]. Available: http://standards.ieee.org/findstds/standard/802.11-2012.html

[10] S. Jana and S. K. Kasera, "On Fast and Accurate Detection of Unauthorized Wireless Access Points Using Clock Skews," in *14th ACM International Conference on Mobile Computing and Networking (MobiCom '08)*, 2008.

[11] T. Kim, H. Park, H. Jung, and H. Lee, "Online Detection of Fake Access Points Using Received Signal Strengths," in *75th IEEE Vehicular Technology Conference (VTC Spring 2012)*, 2012.

[12] F. Lanze, A. Panchenko, B. Braatz, and T. Engel, "Letting the Puss in Boots Sweat: Detecting Fake Access Points using Dependency of Clock Skews on Temperature," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (AsiaCCS 2014)*, 2014.

[13] F. Lanze, A. Panchenko, I. Ponce-Alcaide, and T. Engel, "Undesired relatives: Protection mechanisms against the evil twin attack in ieee 802.11," in *10th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet 2014)*, 2014.

[14] L. Ma, A. Y. Teymorian, and X. Cheng, "A Hybrid Rogue Access Point Protection Framework for Commodity Wi-Fi Networks," in *27th Conference on Computer Communications (INFOCOM 2008)*, 2008.

[15] D. Mónica and C. Ribeiro, "WiFiHop - Mitigating the Evil Twin Attack Through Multi-hop Detection," in *16th European Conference on Research in Computer Security (ESORICS'11)*, 2011.

[16] C. Neumann, O. Heen, and S. Onno, "An Empirical Study of Passive 802.11 Device Fingerprinting," in *32nd International Conference on Distributed Computing Systems Workshops (ICDCS 2012 Workshops)*, 2012.

[17] N. T. Nguyen, G. Zheng, Z. Han, and R. Zheng, "Device fingerprinting to enhance wireless security using nonparametric Bayesian method," in *30th IEEE International Conference on Computer Communications (INFOCOM 2011)*, 2011.

[18] B. Sieka, "Active Fingerprinting of 802.11 Device by Timing Analysis," in *3rd IEEE Consumer Communications and Networking Conference (CCNC 2006)*, 2006.

[19] Y. Song, C. Yang, and G. Gu, "Who Is Peeping at Your Passwords at Starbucks? - To Catch an Evil Twin Access Point," in *40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)*, Chicago, IL, USA, 2010.

[20] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness," in *18th USENIX Security Symposium (SSYM '09)*, 2009.