EvilDirect: A New Wi-Fi Direct Hijacking Attack and Countermeasures

Ala' Altaweel, Radu Stoleru and Guofei Gu
Department of Computer Science & Engineering, Texas A&M University
{altaweel, stoleru, guofei}@cse.tamu.edu

***Abstract*—In this paper, we first show that Group Owner (GO) devices in Wi-Fi Direct are vulnerable to the *EvilDirect* attack. In the EvilDirect attack, a rogue GO is set up by an adversary to look like the legitimate GO (with the same MAC address, SSID, and operating channel). The adversary intercepts the clients' invitation requests and accepts them before the legitimate GO. Accordingly, the adversary hijacks the wireless communications between the clients and the legitimate GO. To defend against the EvilDirect attack, we propose the idea of exploiting the received signal strength (RSS) variations on the wireless channel between each client and the legitimate GO. Our solution, *EvilDirectHunter* checks whether the RSS profiles of both the client and the potential GO devices are similar with each other. Both devices incrementally prove this similarity by exchanging challenge and response packets. EvilDirectHunter is evaluated by implementing it as an Android App, and by modifying the Android kernel code responsible for Wi-Fi Direct in Google Nexus 5 and Samsung Galaxy S2 smartphones. The results show that EvilDirectHunter is able, within seconds, to identify EvilDirect attacks with a high detection rate (100%) while maintaining a low false positive rate (4.5%).**

I. Introduction

Today's smartphones are equipped with a new emerging wireless standard that allows users to establish an adhoc wireless network (without a wireless router) and exchange data among them. The new standard, the Wi-Fi Direct protocol [1], is built upon the IEEE 802.11 infrastructure and it takes a different approach to enhance device-to-device connectivity. Instead of leveraging the adhoc mode of operation between two devices, Wi-Fi Direct enables the devices to form P2P groups by negotiating which device will be the Group Owner (GO) and which devices will be the clients. Wi-Fi Direct is used for data sharing, video streaming, mobile printing, and gaming. The shipments [2] of Wi-Fi Direct devices (tablets, smartphones, and smart TVs) reached around 1,700 million in 2016, and it is predicted to reach three billion in 2019.

In order to establish a secure connection between two devices, Wi-Fi Direct implements the in-band mode of Wi-Fi Protected Setup (WPS) protocol [3]. There are two methods for the in-band mode: PIN and Push-Button methods. In the PIN method, a device password, which is obtained from the GO and entered into the client using a keypad, is used to perform a Diffie-Hellman (D-H) key exchange to guarantee the authentication between the devices. The PIN method vulnerabilities to online [4] and offline [5] brute force attacks were addressed by previous works [6] [7]. In the Push-Button method, the client invites the GO to start the WPS protocol and waits for its acceptance. The GO's user acceptance is used for authentication and to perform a D-H key exchange.

One major security concern and research challenge for the Push-Button method is the vulnerability of the GO to spoofing. We refer to this as *EvilDirect* attack. An EvilDirect is a rogue GO that operates on the same channel as the legitimate GO and has the same MAC address and SSID. It is set up by an adversary, who accepts the clients invitation requests before the legitimate GO, to hijack the wireless communications between the clients and the legitimate GO. The best areas for EvilDirect attacker are public, indoor areas. These areas can be either *dynamic environments* (mobile devices, and/or there are mobile intermediate objects in the environment, e.g., airport lounges, cafes, restaurants, or student community areas), or *static environments* (static devices, and there are few mobile intermediate objects in the environment, e.g., libraries). The EvilDirect attack can be successfully launched because GOs in Wi-Fi Direct have no form of identification except their MAC addresses and SSIDs, which can easily be impersonated. The Push-Button method specifications (pages 78-83 of [3]) state that the client must complete a scan of all 802.11 channels that it supports to discover if any other nearby GOs are in Push-Button mode. If the client discovers more than one GO in Push-Button mode, it must abort its connection attempt and signal an error to the user. However, we successfully launched EvilDirect attack due to the fact that two GOs (with the same MAC address, SSID, and operating channel) are indistinguishable by the clients and treated as a single GO.

Many approaches have been proposed to detect spoofing in wireless networks. [8] [9] [10] detect rogue APs by differentiating (and comparing with an authorized list) the traffic between the wired and wireless connections. Others like [11] [12] [13] monitor Radio Frequency airwaves collected at routers and compare that with a known authorized list. The above methods require a network administrator (not available in Wi-Fi Direct) and authorization lists to check if a client uses a rogue AP. The user oriented approaches [14] [15] detect the rogue APs at the client side by identifying differences in the number of wireless hops (one hop from two hops). These approaches cannot be applied to Wi-Fi Direct since the clients and the GO are connected by a single hop link. Many wireless intrusion detection systems (WIDS) [16] [17] [18], which require a training phase and use sniffers, have been proposed to detect session hijacking attacks in wireless networks. These WIDS are based on behavioral analysis to detect deviations from normal behaviors. However, WIDS are not always avail-

able for Wi-Fi Direct devices. As a result, EvilDirect is still an open research problem for the Push-Button method.

To address the aforementioned research challenge for the *dynamic environments*, we propose to detect the EvilDirect attack by employing the inherent randomness in the wireless channel between the clients and the GO. The properties [19] of the radio channel are unique to the locations of any two devices because: a) The multi-path properties of the channel at any point in time are identical in both directions of the link; b) The temporal variations in the channel (i.e., the multi-path channel changes due to motion of people in the environment). As a result, the adversary will measure a different and uncorrelated radio channel due to the following reasons: a) He cannot be very close to the legitimate GO or the client (i.e., closer than one half of the wavelength, λ); b) He cannot restrict other movements in the area (passengers, customers, etc.). Most of current wireless cards of smartphones are able to measure the received signal strength (RSS). Thus, we exploit this capability in our detection. Our proposed protocol, EvilDirectHunter, is an interactive protocol that executes between each client in the P2P group and the GO. Each client records an RSS profile for the legitimate GO, which in turn records a profile for each client in the P2P group. EvilDirectHunter checks whether the RSS profiles of both the client and potential GO devices are similar to each other by exchanging challenge and response packets. For *static environments*, due to the lack of randomness in the wireless channel, there are low variations in the RSS profiles. These low RSS variations are not sufficient to detect the EvilDirect attacks. Accordingly, our EvilDirectHunter executes an additional detection phase, which is designed for the static environments. This phase is based on Multi-Dimensional Scaling (MDS) algorithm [20] and involves a cooperation among all clients in the P2P group. Moreover, this phase requires that each client calculates the average of the RSS values of the last five packets that were overheard by it when any other client exchanged packets with the legitimate GO.

The contributions of this paper are as follows: a) We demonstrate a successful EvilDirect attack against Wi-Fi Direct GOs; b) We design a new protocol, EvilDirectHunter, to detect the EvilDirect attacks in both dynamic and static environments; c) We demonstrate the feasibility of EvilDirectHunter through a real implementation on Google Nexus 5 and Samsung Galaxy S2 smartphones; d) We prove the effectiveness of EvilDirectHunter by showing that it identifies EvilDirect attacks with a high detection rate while maintaining a low false positive rate.

II. Wi-Fi Direct Background

Wi-Fi Direct's devices have dynamic roles [1] (the role of GO and the role of client). These devices form P2P groups, which are equivalent in terms of functionalities to the Wi-Fi infrastructure networks. In any P2P group, P2P GO implements the AP-like functionality, and the other devices are P2P clients. There are three group formation mechanisms [1] to establish a P2P group: Standard, Autonomous (which are our focus in this paper), and Persistent. Each mechanism has three main phases: *Device Discovery*, *Service Discovery*, and *Wireless Protected Setup Provisioning*.

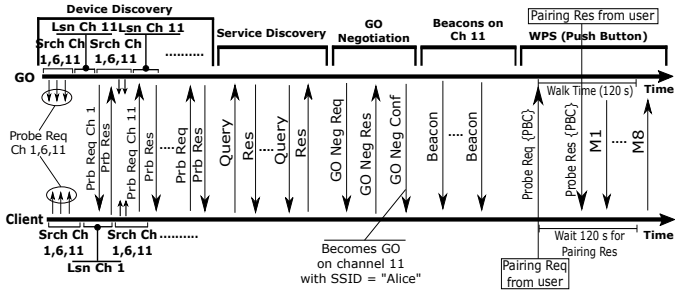


Fig. 1: Frame exchange sequence for Standard mechanism.

The P2P devices in the Standard formation mechanism start by executing a "discovery algorithm" as shown in Figure 1. First, each P2P device selects one of the channels (i.e., channels 1, 6, or 11 in the 2.4 GHz band) as its listening channel. Second, a P2P device alternates between the search and listen states. In the search state, the device performs active scanning by sending probe requests in each channel. In the listen state, the device listens for probe requests in its listening channel to respond with probe responses. After the two P2P devices find each other, they start the Service Discovery phase.

The Service Discovery phase in the Standard formation mechanism (shown in Figure 1) is an additional feature for Wi-Fi Direct devices, which was not present in the traditional Wi-Fi networks (the only service in which clients are interested in, is Internet connectivity). Before establishing a P2P group, P2P devices exchange queries to discover their available services and, based on this, decide whether to continue the group formation or not. Service discovery queries are generated by a higher layer protocol (UPnP or Bonjour [21]), and employing the link layer Generic Advertisement Protocol (GAS) specified by 802.11u [22]. GAS is a layer two query/response protocol implemented through the use of public action frames, that allows two non-associated devices to exchange queries belonging to a higher layer protocol.

The Standard formation mechanism has an additional phase, the GO Negotiation (shown in Figure 1), which is implemented using three-way handshake messages (Request, Response, and Confirmation). These messages enable the two devices to decide which one will be the GO and on which channel the P2P group will operate. Each device sends a numerical parameter (Intent Value (IV)) and a tiebreaker bit within the handshake messages. The device with the highest IV becomes the GO. The tiebreaker bit prevents conflicts when two devices declare the same IV. In Figure 1, the upper device wins the GO Negotiation, and selects channel 11 for its P2P group with SSID="Alice" (set by the user). Then, this GO starts to beacon on channel 11 and waits for new clients to join its group (the new clients will not execute the GO Negotiation phase).

The Wireless Protected Setup (WPS) Provisioning [3] phase is implemented by the Standard formation mechanism, as shown in Figure 1. WPS aims to achieve a secure connection between P2P clients and P2P GO. WPS has two modes of operation: in-band mode and out-of-band mode. There are two methods for the in-band mode: Push-Button method and PIN method. Both methods are used for authentication and to establish Diffie-Hellman keys between the client and the GO.

In the Push-Button method, which is our focus in this paper, the client invites the GO to start the WPS protocol and waits

for its acceptance. As shown in Figure 1, the P2P client's user presses a logical button to invite his friend's GO device (with SSID="Alice") to start the WPS protocol. The client device includes its mode (Push-Button Configuration (PBC)) in the invitation to notify the GO to enable it. The GO's user either accepts or declines the invitation by pressing a logical button within 120 seconds (known as Walk Time). In case of acceptance, the two devices execute the eight registration protocol messages (M_1 to M_8 in Figure 1) of the WPS protocol and connect to each other (page 33 of [3]).

The Autonomous formation mechanism [1] enables the P2P device's user to immediately set his device as a GO and create a P2P group (with specific SSID and channel). Then, this GO starts to beacon on the specified channel. Other devices can discover the P2P group using traditional scanning mechanisms and directly proceed with the Service Discovery and WPS Provisioning phases (no GO Negotiation phase is required).

III. Motivation & State of the Art

We successfully launched an EvilDirect attack against GO devices due to the fact that two GOs with the same MAC address and SSID, that operate on the same channel are indistinguishable by the clients and treated as a single GO. In order to accomplish this attack, the attacker needs a smartphone/laptop running Ubuntu. The first step for the attacker is to discover (using Apps like WiFiScanner [23]) the MAC address, SSID, and operating channel of the legitimate GO. The second step for the attacker is to set his EvilDirect GO to look like the legitimate GO. On a low end notebook, we configured the wireless interface in monitor mode. Then we used the Airbase-ng attack tool to create the EvilDirect GO (with the same MAC address, SSID, and channel as the legitimate GO (shown in Figure 2)). When the client's user tries to connect to the legitimate GO, his device invites the legitimate GO to start the WPS protocol and waits for its acceptance for 120 seconds, as shown in Figure 2. The EvilDirect GO receives this invitation too, as shown in Figure 2. Before the legitimate GO's user accepts this invitation by pressing a logical button, the EvilDirect GO can reply with an acceptance to the client's invitation (with the same MAC address and SSID as the legitimate GO). Then, the EvilDirect GO executes the eight registration protocol messages of the WPS protocol with the client and hijacks its wireless communication.

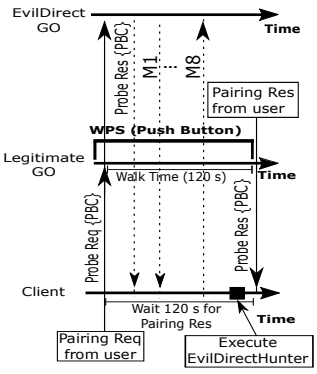


Fig. 2: EvilDirect attack.

In Wi-Fi infrastructure networks, the evil twin AP attack occurs mainly in free public Wi-Fi areas (airport lounges, cafes). The existing rogue AP detection solutions can be classified into two categories: network administrator oriented, which are designed for a wireless network administrator to perform access control policies and authorization for the APs and clients, and user oriented (client-side solutions).

The network administrator solutions can be classified into two approaches. The first approaches proposed in [24] [8] [25] [26] [27] [28] [29] [9] [30] [10] detect the attacker by differentiating the traffic between the wired and wireless connections. If an unauthorized client uses a wireless network while it is not authorized to do so (by comparing with an authorized list), the AP attached to this client is classified as a rogue AP. [25] employs the round trip time (RTT) between the user and the DNS server to determine whether an AP is legitimate or not. [27] and [30] use packets inter arrival time (IAT) and RTT to distinguish between Ethernet and wireless clients. [28] assumes that the mean and variance of packets IAT is more random for a network path that has a wireless link as compared to a path that has only wired links. [8] and [9] propose algorithms that exploit the fundamental properties of the 802.11 CSMA/CA MAC protocol and the half duplex nature of wireless channels to differentiate Ethernet and WLAN TCP traffic. The second network administrator oriented approaches proposed in [11] [12] [13] [31] [32] [33] work by monitoring Radio Frequency (RF) airwaves and additional information collected at routers and compare that with a known authorized list. [11] utilizes the fact that different APs usually have different clock skews to detect unauthorized APs. [12] locates suspicious APs by scanning RF airwaves from the Intranet and then compares that with specific fingerprints of the RF (with an authorized list). [13] uses desktop machines in monitoring by attaching them to USB-based wireless adapters. [33] uses sensors instead of sniffers to scan the RF. The user oriented solutions proposed in [14] [15] detect an evil twin AP attack at the client side (without any additional support from the network administrator). CETAD [14] is designed based on the idea that the public IP address, ISP, and RTT values of packets traveling through legitimate APs are similar (the same ISP), but they are different for a legitimate AP and an evil twin AP. [15] proposes two algorithms to detect an evil twin AP from the client end based on server IAT. The two algorithms need a remote server within the LAN with an installed software to measure the server IAT.

The current solutions to the evil twin AP attack in Wi-Fi infrastructure networks are not applicable to Wi-Fi Direct due to the following reasons: a) There are no network administrators in Wi-Fi Direct networks (authorized list approaches are not working); b) Differentiating the traffic at the wired and wireless connections is not possible (no wired connection in Wi-Fi Direct networks); c) Intrusion detection systems, sniffers and sensors are not always available for Wi-Fi Direct devices; d) [14] and [15] cannot be used because they depend on differentiating the wireless hops (one or two hops), however, the clients and the GO are connected by a single hop link.

IV. EvilDirectHunter

This section presents our main idea, the adversary model, the design of EvilDirectHunter protocol, and the tuning of its parameters.

A. Main Idea

As shown in Figure 3(a), EvilDirectHunter is based on the idea that while the client and the legitimate GO execute

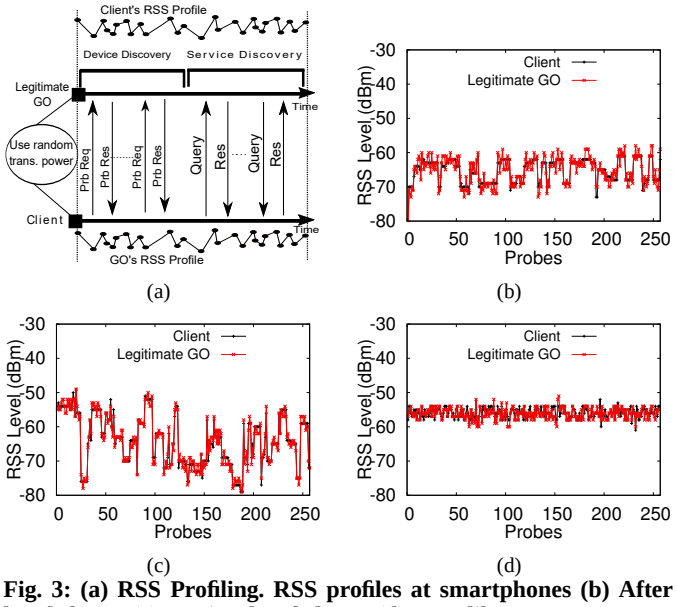


Fig. 3: (a) RSS Profiling. RSS profiles at smartphones (b) After lunch hour. (c) During lunch hour. (d) In a library.

the Device Discovery and Service Discovery phases in both Standard and Autonomous group formation mechanisms, the client records an RSS profile with say $2k$ samples, $k \in \mathbb{N}$ (i.e., $[rss_1, rss_2, ...rss_k, rss_{k+1}, ...rss_{2k}]$) for the legitimate GO, which in turn records a profile for each client in the P2P group with the same number of samples. Moreover, each client calculates the average of the RSS values of the last five packets that were overheard by it when any other client exchanged packets with the legitimate GO.

In order to validate whether the RSS profile is suitable to differentiate legitimate GO and EvilDirect GO for dynamic and static environments, we performed three experiments using two Google Nexus 5 smartphones and recorded 256 RSS readings of these smartphones. The distance between the two devices in these experiments, equals 10 m. The goals of these experiments are to examine the *randomness* (i.e., the variation window), and the *degree of similarity* (i.e., the Mean Squared Error (MSE) = $\frac{1}{256}\sum_{i=1}^{256}[RSSClient_i - RSSGO_i]^2$) of the two devices' RSS profiles (i is the i^{th} RSS reading). With MSE closer to zero indicating a higher degree of similarity.

The first two experiments (with results shown in Figures 3(b) and 3(c)) were conducted in a dynamic indoor environment (a cafeteria during and after a busy lunch hour (12:00 PM - 1:00 PM)). The third experiment (Figure 3(d)) was conducted in a static indoor environment (a library building on a weekday evening). As shown in Figures 3(b) and 3(c), the variation window of the RSS profile (i.e., the randomness of the channel) increases when we have more moving intermediate objects in the environment between the client and the legitimate GO. Moreover, the mobility of the intermediate objects helps in achieving a higher degree of similarity of the RSS profiles (the MSE of the two devices' RSS profiles for Figures 3(b) and 3(c) are: 3.43 and 1.81, respectively). This mobility improves the degree of similarity due to the fact that it dominates the RSS profile variation more than the random thermal noise and different interference sources, which affect the client and the legitimate GO differentially. However, as shown in Figure 3(d), there are no noticeable variations in the channel (low Shannon entropy of the RSS profile), and the MSE of the two devices' RSS profiles = 13.36 indicating a low degree of similarity (the variations in a static channel are generated by thermal effects and hardware imperfections).

B. Adversary Model

Our adversary model assumes that a malicious attacker, Eve, is able to set up an EvilDirect GO with the same MAC address and SSID as the legitimate GO, and operates on the same channel. The best areas for Eve are public, indoor (airport lounges, cafes, restaurants, or student community areas). Eve is free to move in the area but she cannot be very close to the legitimate GO or the clients (we only require that she is more than 1 m away from them). Moreover, Eve is unable to restrict other movements in the channel (passengers in airport lounges, customers/waiters in cafes, students in student community areas). We assume that Eve knows EvilDirectHunter protocol, but she has no access to the RSS profiles of the devices. Eve can listen to all communication between the clients and the legitimate GO. Also, she can measure both the channels between herself and the clients and between herself and the legitimate GO at the same time when the clients and the legitimate GO execute the Device Discovery and Service Discovery phases and build each other's RSS profile. We assume that Eve is not interested in launching a denial-of-service attack (DoS) or jamming the communication channel between the clients and the legitimate GO (which is the same channel for her EvilDirect GO). Also, Eve is not equipped with full-duplex radio transceivers or directional antennas that enable the reception of a signal from the legitimate GO and jamming of the same signal at the clients, or vice versa. Moreover, we assume that Eve doesn't modify any messages exchanged between the clients and the legitimate GO when they execute the Device Discovery and Service Discovery phases, otherwise, it might raise suspicion and be detected. However, Eve can launch a replay attack as described below. Consequently, our solution doesn't address all man-in-the-middle attacks. Based on the assumptions described above, Eve can launch the following types of attacks:

1) **Physical proximity attack**: Eve aims to obtain the same RSS profile of the client or the legitimate GO in order to pass the detection of EvilDirectHunter. Eve tries to (physically) get close to the victims to have the same RSS readings.

2) **Predictable channel attack**: Another way for Eve to obtain the same RSS profile of the client or the legitimate GO is to predict the RSS readings at their physical locations to fake her RSS readings in dynamic and static environments.

3) $\oplus$ **result guessing attack**: The client and the legitimate GO divide the RSS profile that they build for each other into two subsets ($Rss_I = [rss_1, rss_2 ..rss_k]$, $Rss_{II} = [rss_{k+1}, rss_{k+2} ..rss_{2k}]$). Then, they calculate the (Euclidean distance)^2, denoted as $\oplus$, between these two subsets. $\oplus$ measures the similarity between the two RSS profiles. Eve is able to launch an online/offline guessing attack against the result of $\oplus$ to pretend to be the legitimate GO.

4) **Client spoofing attack**: Eve can try to obtain the results of $\oplus$ by pretending to be one of clients. Before launching her

TABLE I: NOTATIONS AND DEFINITIONS

C: Client device.
n: Total number of devices in the P2P group (clients and GO).
$\boldsymbol{\eta}$: max $\boldsymbol{j} \in \mathbb{N}$, s.t. $\boldsymbol{2^j <= size(RssPrfl)}$.
$\boldsymbol{H(GOprfl)}$: $-\sum_{i=1}^{l} \boldsymbol{p(GOprfl_i) \times log_2 p(GOprfl_i)}$, s.t. $\boldsymbol{l = size(GOprfl)}$.
$\boldsymbol{\varepsilon}$: Threshold of Shannon entropy for the dynamic environments.
$\boldsymbol{\beta}$: Smallest size of RSS subset (i.e., 16 RSS readings).
$\oplus$: (Euclidean distance)^2 between RSS subsets.
$\boldsymbol{\tau}$: Threshold of difference between C and GO $\oplus$ calculations.
$\boldsymbol{D}$: $\boldsymbol{(n \times 2) \times (n \times 2)}$ RSS distance matrix.
$\boldsymbol{Y}$: $\boldsymbol{(n \times 2) \times 2}$ configuration matrix. (i.e., output of MDS).

attack, Eve waits until the client and the legitimate GO build each other's RSS profile. Then, she executes EvilDirectHunter with the legitimate GO to learn the results of $\oplus$ for that client. Finally, Eve launches her attack and passes EvilDirectHunter with that client since she is able to respond to its challenges.

5) **Replay attack**: Eve can overhear the messages between the clients and the legitimate GO (during the Device Discovery and Service Discovery phases) and replay all/some of these messages in order to influence their RSS profiles.

C. EvilDirectHunter Protocol

EvilDirectHunter is an interactive protocol that runs between each client and the GO. The client starts the execution of EvilDirectHunter after the execution of the eight messages of WPS protocol with the potential GO that accepts its invitation, as shown in Figure 2, (i.e., all EvilDirectHunter's messages are encrypted and authenticated using the WPS secret keys). Both devices incrementally prove the mutual knowledge of the RSS profile by exchanging challenge and response packets. Table I presents the notations we use in describing EvilDirectHunter. Essentially, the GO responds to the client's challenge and creates a new challenge to the client in each packet, and vice versa. Algorithms 1 and 2 present the pseudo code of EvilDirectHunter at the client and GO devices, respectively.

1) Phase I of EvilDirectHunter for dynamic and static environments: the client reads the first 2^η of the GO's RSS profile ($GOprfl$) (e.g., for 290 RSS readings, it reads the first 256 readings). For the first pass, $pass\# = 1$, the client creates a challenge packet by dividing the $GOprfl$ into two subsets using the Pick2RSS function presented in Algorithm 3. First, Pick2RSS divides the $GOprfl$ into $total$ subsets based on the value of the $pass\#$ (i.e., 2 subsets each of size 128 for the 256 readings). Second, it picks two random subsets, which were not selected before as challenges. These two random subsets are tagged so that they will not be picked again for the next challenges (line 8 of Algorithm 3). Pick2RSS returns $\langle 1, Rss_1, Rss_2 \rangle$ for the first pass (Rss_1 and Rss_2 contain the first and last 128 RSS readings of the $GOprfl$, respectively).

The client calculates the $\oplus$ metric (defined in Table I) between Rss_1 and Rss_2 subsets, and stores the result in $challenge_C$ (line 4 of Algorithm 1). Then, it sends its challenge packet, $\langle pass\#, id_I, id_{II}, null \rangle$, which contains the indices of its challenge to the GO, and waits for its response and challenge (lines 5 & 7 of Algorithm 1). For $pass\# = 1$, there is no challenge for the client ($response_C$ is $null$).

Algorithm 1 EvilDirectHunter at each Client

1: Read the first 2^η of $GOprfl$
2: $pass\# \leftarrow 1$
3: $\langle pass\#, Rss_{id_I}, Rss_{id_{II}} \rangle \leftarrow Pick2RSS(pass\#, GOprfl, 2^\eta)$
4: $challenge_C = Rss_{id_I} \oplus Rss_{id_{II}}$
5: send $\langle pass\#, id_I, id_{II}, null \rangle$ to GO
6: **while** $true$ **do**
7: wait $\langle pass\#, id_I, id_{II}, response_{GO} \rangle$ from GO
8: **if** $|challenge_C - response_{GO}| <= \tau$ **then**
9: **if** $id_I \neq id_{II} \neq -1$ **then**
10: Rss_I =GetSubset ($pass\#, GOprfl, 2^\eta, id_I$)
11: Rss_{II} =GetSubset ($pass\#, GOprfl, 2^\eta, id_{II}$)
12: $response_C = Rss_I \oplus Rss_{II}$
13: $\langle pass\#, Rss_{id_I}, Rss_{id_{II}} \rangle \leftarrow Pick2RSS(pass\#, GOprfl, 2^\eta)$
14: **if** $pass\# \neq null$ **then**
15: $challenge_C = Rss_{id_I} \oplus Rss_{id_{II}}$
16: send $\langle pass\#, id_I, id_{II}, response_C \rangle$ to GO
17: **else**
18: $GOprflEntropy \leftarrow H(GOprfl)$
19: **if** $GOprflEntropy > \varepsilon$ **then**
20: **Legitimate GO, Stop**
21: **else**
22: Broadcast V
23: $D \leftarrow$ get V's of other clients
24: $Y \leftarrow$ MDS ($|D|, 2$)
25: $\left\langle d(R_{GO}, R_{\widehat{GO}}), Avg_{i:1 \to n}[d(R_{C_i}, R_{\widehat{C_i}})] \right\rangle \leftarrow Y$
26: **if** $d(R_{GO}, R_{\widehat{GO}}) > Avg_{i:1 \to n}[d(R_{C_i}, R_{\widehat{C_i}})]$ **then**
27: **EvilDirect GO, Stop**
28: **else**
29: **Legitimate GO, Stop**
30: **else**
31: **EvilDirect GO, Stop**

Algorithm 2 EvilDirectHunter at the GO

1: Read the first 2^η of $Cprfl$
2: **while** $true$ **do**
3: wait $\langle pass\#, id_I, id_{II}, response_C \rangle$ from C
4: **if** $pass\# == 1$ or $|challenge_{GO} - response_C| <= \tau$ **then**
5: Rss_I =GetSubset ($pass\#, Cprfl, 2^\eta, id_I$)
6: Rss_{II} =GetSubset ($pass\#, Cprfl, 2^\eta, id_{II}$)
7: $response_{GO} = Rss_I \oplus Rss_{II}$
8: $\langle pass\#, Rss_{id_I}, Rss_{id_{II}} \rangle \leftarrow Pick2RSS(pass\#, Cprfl, 2^\eta)$
9: **if** $pass\# \neq null$ **then**
10: $challenge_{GO} = Rss_{id_I} \oplus Rss_{id_{II}}$
11: **else**
12: $id_I \leftarrow id_{II} \leftarrow -1$
13: send $\langle pass\#, id_I, id_{II}, response_{GO} \rangle$ to C
14: **else**
15: **Illegitimate Client, Stop**

The GO waits for the client packet as shown in line 3 of Algorithm 2 after it reads the first 2^η of the client's RSS profile ($Cprfl$). For the first pass, the GO responds to the client's challenge by getting the two RSS subsets of $Cprfl$, which correspond to id_I and id_{II} (indices of the client's challenge to the GO). The GO uses the GetSubset function (Algorithm 4) to get these two RSS subsets (lines 5 & 6 of Algorithm 2). Then, the GO calculates $\oplus$ between these two subsets (line 7 of Algorithm 2) to prepare its response. Moreover, the GO prepares its challenge using Pick2RSS function. Since the two 128 RSS subsets of $Cprfl$ have been used as challenges by the client, Pick2RSS increases $pass\#$ by one. Accordingly, $Cprfl$ is divided into four subsets (each of size 64 for the 256 readings). Pick2RSS randomly picks two subsets out of them to challenge the client. The GO sends $\langle pass\#, id_I, id_{II}, response_{GO} \rangle$ (line 13 of Algorithm 2), which contains the $pass\#$ and indices for its new challenge,

Algorithm 3 Pick Two Random RSS Subsets

1: **function** Pick2RSS ($pass\#, rssProfile, 2^\eta$)
2: $subsetSize \leftarrow 2^\eta / 2^{pass\#}$
3: $total \leftarrow 2^\eta / subsetSize$
4: Divide $rssProfile$ evenly into $\{rss_1, rss_2 ... rss_{total}\}$
5: $unused \leftarrow \{rss_1, rss_2 ... rss_{total}\} - rss_i$, s.t. rss_i is tagged
6: **if** $size(unused) >= 2$ **then**
7: Pick $\{Rss_I, Rss_{II}\}$ randomly from $unused$
8: Tag Rss_I and Rss_{II} as used subsets
9: Find i & $j \in \mathbb{N}$, s.t. Rss_I & Rss_{II} are the i_{th} & j_{th} subsets of $\{rss_1, rss_2 ... rss_{total}\}$, respectively
10: **return** $\langle pass\#, Rss_i, Rss_j \rangle$
11: **else**
12: **if** $subsetSize >= \beta$ **then**
13: $pass\#$++
14: **goto 2**
15: **else**
16: **return** $null$

Algorithm 4 Get the RSS Subset

1: **function** GetSubset ($pass\#, rssProfile, 2^\eta, index$)
2: $subsetSize \leftarrow 2^\eta / 2^{pass\#}$
3: $total \leftarrow 2^\eta / subsetSize$
4: Divide $rssProfile$ evenly into $\{rss_1, rss_2 ... rss_{total}\}$
5: Find the rss_i in $\{rss_1, rss_2 ... rss_{total}\}$, s.t. $i = index$
6: Tag rss_i as a used subset
7: **return** rss_i

and the response to the client's previous challenge.

Both devices follow the protocol by responding to other device's challenge and preparing its new challenge as long as $|challenge_{C/GO} - response_{GO/C}| <= \tau$ (lines 8 & 4 of Algorithms 1 and 2, respectively). Otherwise, either the client detects the GO as an EvilDirect GO (line 31 of Algorithm 1), or the GO stops the protocol (line 15 of Algorithm 2) to prevent the client spoofing attack (which will be discussed in Section VI-D). By increasing the $pass\#$ in Pick2RSS function, the size of the RSS subsets that are being challenged between the two devices becomes smaller. The goal of decreasing the subsets size is to ensure the similarity between the two RSS profiles even for the small portions.

If the two devices successfully execute the protocol (the Pick2RSS function returns $null$ since the $subsetSize < \beta$), the client has to check if the channel is dynamic or static (lines 18 and 19 of Algorithm 1) by calculating the Shannon entropy of $GOprfl$. If $GOprflEntropy$ is greater than ε, this indicates a dynamic channel. Accordingly, EvilDirectHunter protocol stops and establishes the P2P group (line 20 of Algorithm 1). Otherwise, an additional detection phase is executed for the static environments.

2) Phase II of EvilDirectHunter for static environments: The solution we propose for detecting EvilDirect GO in static environments makes use of Multi-Dimensional Scaling (MDS). MDS [20] is a class of statistical techniques used to discover relationships in a set of data. The basic idea is that given n objects and a numerical $n \times n$ proximity matrix representing inter-object dissimilarities, an equivalent representation of n points in m-dimensional space can be found whose inter-point distances are proportional to the similarities. MDS algorithm has been used for wormhole localization in mobile ad hoc networks [34]. The proximity matrix in [34] represents the travel distances of the mobile nodes. Due to the fact that there are low variations in the RSS levels in static environments (Figure 3(d)), and that the RSS level decreases 6.02 dBm each time the distance from the source is doubled [35], we decided to use the RSS levels between the clients and the legitimate GO to build their proximity matrix.

Each client broadcasts a V vector (line 22 of Algorithm 1), which contains the average of RSS values of the last five packets that were overheard by it when other clients exchanged packets with the GO before and after each client receives its invitation acceptance (i.e., Probe Res {PBC}) from the GO. For example, if we have two clients (A and B), client A's $V = [RSS^A_B, RSS^A_{\widehat{B}}, RSS^A_{GO}, RSS^A_{\widehat{GO}}]$ (where $\widehat{.}$ represents the device after receiving the invitation acceptance). RSS^A_B is the average of the RSS values of the last five packets that were overheard by client A when client B exchanged packets with the legitimate GO before it receives the invitation acceptance. $RSS^A_{\widehat{B}}$ is the average of the RSS values of the last five packets that were overheard by client A when client B exchanged packets with the GO after it receives the invitation acceptance. RSS^A_{GO} is the average of the RSS values of the last five packets that were received by client A from the GO before it receives its invitation acceptance. $RSS^A_{\widehat{GO}}$ is the average of the RSS values of the last five packets that were received by client A after it receives its invitation acceptance. Client B's $V = [RSS^B_A, RSS^B_{\widehat{A}}, RSS^B_{GO}, RSS^B_{\widehat{GO}}]$. Each client builds a matrix D (symmetric RSS distance matrix) based on its V and the received V's from other clients. For our example above, the D matrix for client A is shown in Figure 4. N indicates the absence of the RSS value (e.g., $RSS^A_{\widehat{A}}$, $RSS^A_{\widehat{B}}$, $RSS^A_{\widehat{GO}}$). We use R instead of RSS, for condensed notation. The rows represent the RSS distance vectors for devices $A, B, \widehat{A}, \widehat{B}, GO, \widehat{GO}$, respectively. These vectors are to devices $A, B, \widehat{A}, \widehat{B}, GO, \widehat{GO}$, respectively ($D$'s columns).

The absolute value of D matrix, $|D|$, is passed to the MDS algorithm (line 24 of Algorithm 1). The steps of the MDS algorithm are presented in [20]. The output of MDS is Y, the set of 2-dimensional coordinates that describes the similarities between each device's RSS values before and after it receives its invitation acceptance from the GO. The EvilDirect GO can be detected due to the high difference between his RSS values (received/overheard by all clients after they receive the acceptance for their invitations), and the legitimate GO's RSS values (received/overheard by all clients before they receive the acceptance for their invitations). This high difference is due to the fact that the EvilDirect GO has a different physical location corresponding to all clients in the P2P group compared to the legitimate GO physical location. On the other hand, since the clients are static, and there are few intermediate mobile objects in static environments, the differences between clients' RSS values (before and after they receive the invitation acceptance from the GO) are low. Accordingly, any client can detect the EvilDirect GO attack

$$\begin{pmatrix} 0 & R^A_B & \text{N} & \text{N} & R^A_{GO} & \text{N} \\ & 0 & \text{N} & \text{N} & R^B_{GO} & \text{N} \\ & & 0 & R^{\widehat{A}}_{\widehat{B}} & \text{N} & R^{\widehat{A}}_{\widehat{GO}} \\ & & & 0 & \text{N} & R^{\widehat{B}}_{\widehat{GO}} \\ & & & & 0 & \text{N} \\ & & & & & 0 \end{pmatrix}$$

Fig. 4: D matrix

by comparing the difference between the GO's RSS values (before and after receiving the invitation acceptance) to the average of differences between all clients' RSS values (before and after they receive the acceptance for their invitations).

For any client, $d(R_{GO}, R_{\widehat{GO}})$ is the Euclidean distance between the two RSS values of the GO before and after it receives its invitation acceptance (line 25 of Algorithm 1). $Avg_{i:1\rightarrow n}[d(R_{C_i}, \widehat{R_{C_i}})]$ is the average of Euclidean distances of all clients' RSS values before and after they receive the acceptance for their invitations. The client compares $d(R_{GO}, R_{\widehat{GO}})$ and $Avg_{i:1\rightarrow n}[d(R_{C_i}, \widehat{R_{C_i}})]$ (line 26 of Algorithm 1) to detect the EvilDirect attack. In the normal scenarios, $d(R_{GO}, R_{\widehat{GO}}) \leq Avg_{i:1\rightarrow n}[d(R_{C_i}, \widehat{R_{C_i}})]$, which indicates a slight change in the GO's RSS values.

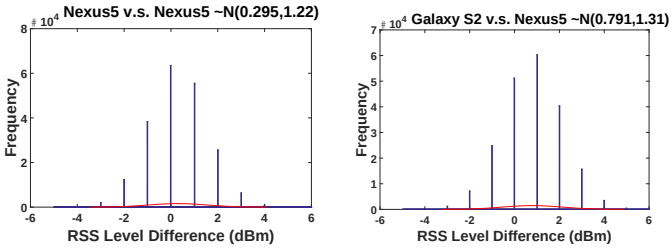
D. EvilDirectHunter Parameter Tuning

This section presents the measurement and tuning of ε and τ thresholds, respectively.

1) Measuring the threshold parameter ε: We repeated each experiment presented in Figures 3(b) and 3(c) 267 times. We aim to experimentally investigate the suitable value of ε, which is used to distinguish between static and dynamic environments. Accordingly, we measured the average value of $GOprflEntropy$ of these 534 experiments and we set ε to 4.53. We acknowledge that the value of ε might be different for other environments, and it requires further study.

2) Tuning the threshold parameter τ: τ is used by the client to check if the GO is a legitimate or an EvilDirect device (line 8 of Algorithm 1). We aim to find the best value of τ for all passes when EvilDirectHunter runs on same/different smartphones. We repeated each experiment presented in Figure 3 267 times (total = 801 experiments) to generate 205,056 RSS readings using Google Nexus 5 smartphone for the client and the GO. Each experiment generates 256 RSS readings for the GO, RSS_{GO}, and the client, RSS_{Client}. We assume and validate in Section VI-C that the distribution of any RSS reading (say m) of the client is $rss_m \sim \mathcal{N}(\mu, \sigma^2)$. We calculated [$RSS_{GO}$ - RSS_{Client}] for each experiment to create a vector of 256 RSS level differences (dBm). We plotted the histogram for the 801 [RSS_{GO} - RSS_{Client}] vectors that were generated from these 801 experiments. Figure 5(a) shows the histogram of the 205,056 (801×256) RSS level differences. It's clear that the difference between the GO's and client's RSS readings (RSS_{GO} - RSS_{Client}) is distributed $\sim \mathcal{N}(\mu, \sigma^2)$.

In order to calculate the best value of τ for all passes when EvilDirectHunter executes between two Google Nexus 5 smartphones, we run a Monte Carlo simulation with the following steps. First, we generated 256 RSS values using the normal distribution (i.e., which we assume and validate in Section VI-C) for the client device (RSS_{Client}). Second, we generated 256 RSS values for the GO, RSS_{GO} = RSS_{Client} + x. x is a random number generated from the normal distribution presented in Figure 5(a). Third, we calculated the $d^2(Rss_I, Rss_{II})$'s for all passes for both the client and the GO. Fourth, we calculated the differences between the client d^2's and the GO d^2's for all passes. Fifth, we repeated the previous steps 100,000 times, and we calculated the average



(a) (b)

Fig. 5: Histograms of the RSS level difference between (a) Two Nexus 5 smartphones. (b) Galaxy S2 and Nexus 5 smartphones.

of the differences between the client d^2's and the GO d^2's. Table II presents the τ values (dBm) for all passes when EvilDirectHunter executes on Google Nexus 5 smartphones.

We repeated the same 801 experiments using Samsung Galaxy S2 for the client, and Google Nexus 5 for the GO. Figure 5(b) shows the histogram of the 205,056 RSS level differences of these experiments. Moreover, we repeated the same steps of the Monte Carlo simulation for these smartphones. However, we used the normal distribution of Galaxy S2's RSS level (omitted here due to space constraints), and the normal distribution presented in Figure 5(b). Table II shows the τ values (dBm) for all passes when EvilDirectHunter executes on Samsung Galaxy S2 and Google Nexus 5 smartphones. We use the τ values in Table II in our experiments that we will present in Section VII.

TABLE II: τ values (dBm).

Pass#	Nexus5 v.s. Nexus5	S2 v.s. Nexus5
1	809.6	923.8
2	404.9	462.2
3	202.5	231.1
4	101.2	115.6

V. Implementation & Experimental Setup

In order to record the RSS profile for the client and the GO, we downloaded, modified and built the Cyanogenmod [36] Android kernel code for Google Nexus 5 and Samsung Galaxy S2 smartphones. The wireless card drivers of these smartphones report the RSS values as integers from -25 to -100 dBm. The Android kernel code that is responsible for establishing the Wi-Fi Direct connection for these smartphones implements the "discovery algorithm" (illustrated in Section II). We modified the Android kernel code such that each device stores the RSS value of the received probe request/response frames during its listen state. In the original Android kernel code, the amount of time for the search and listen states is randomly distributed between 100 and 300 msec. Since the client and the GO respond with probe responses only during their listen states, we set the duration of the listen and search states to 100 msec to get 10 RSS readings/sec. Each device might be the initiator of the probe request frames (during its search state), or the responder with the probe response frames (during its listen state). The initiator injects the header of the probe request frame with a specific sequence number in order to handle the frame losses and retransmissions. If the probe response frame has the same sequence number as the request frame, the initiator records the RSS value of that frame. Moreover, If the initiator receives two probe response frames with the same sequence number, it records both of them. Also, If the responder receives two probe request frames with the same sequence number, it records both of them.

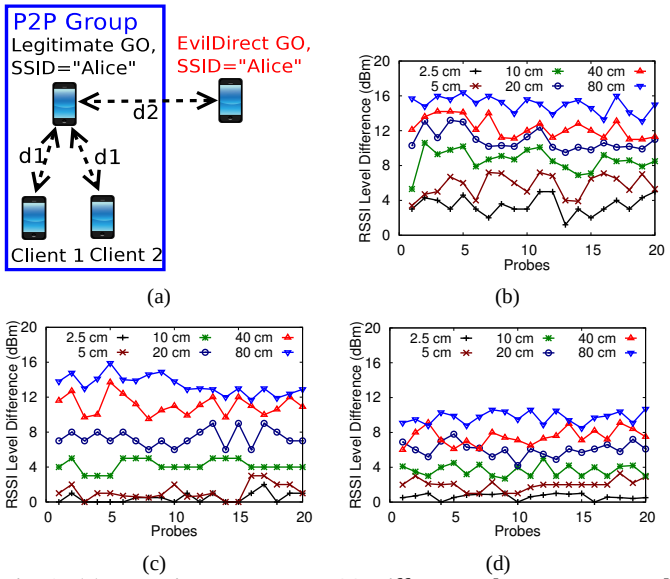


Fig. 6: (a) Experiments setup. RSS Differences between Eve and Legitimate GO for experiments (b) A. (c) B. (d) C.

We implemented the "EvilDirectHunter" Android App, which enables the smartphones' users to detect the EvilDirect GOs.

We performed three experiments using four smartphones in different environments as shown in Figure 6(a). $d1$ (the distance between each client and the legitimate GO) = 10 m. $d2$ is the distance between the legitimate GO and the attacker. In each experiment, both clients execute the Device Discovery and Service Discovery phases with the legitimate GO for 30 seconds. The clients and GO's smartphones record the RSS readings of their last 256 packets. The attacker's smartphone overhears the packets that were sent from each client to the legitimate GO, and records the RSS readings of the last 256 packets to launch his attack. In the following, we describe our experiments: **1) Experiment A**: This experiment was conducted in a crowded cafeteria during a busy lunch hour (12:00 PM - 1:00 PM). There were around 70 customers inside the cafeteria; **2) Experiment B**: This experiment was conducted in a cafeteria after the lunch hour (1:30 PM - 2:30 PM). There were around 25 customers inside the cafeteria; **3) Experiment C**: This experiment was conducted in the study area of a library on a weekday evening (10:00 PM - 11:00 PM). There were 5 students on the study seats.

VI. Security Analysis

A. Physical proximity attack defense

[19] proved that Eve, who is more than half a wavelength away from the client and the legitimate GO, experiences fading channels to them that are statistically independent from the fading between the client and the legitimate GO. In order to investigate the value of $d2$ distance that enables Eve to obtain the same RSS readings as the legitimate GO, we performed experiments A, B, and C while we varied $d2$ distance between 2.5 cm to 80 cm. Eve's smartphone can overhear the packets that were sent from Client 1 (Figure 6(a)) to the legitimate GO, and records the RSS readings of the last 256 packets. We plotted the absolute difference between the RSS readings of the first 20 packets (for illustration purposes) that were received by the legitimate GO from Client 1, and the RSS

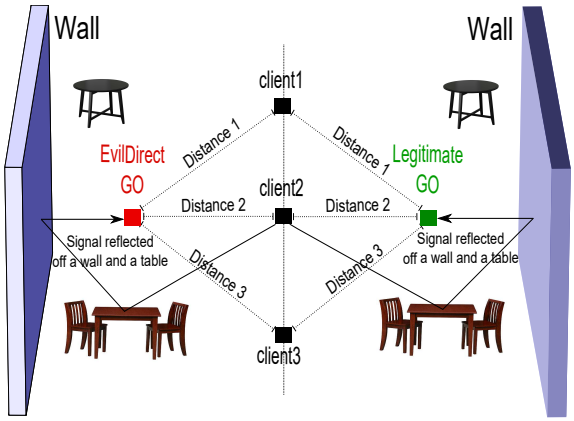


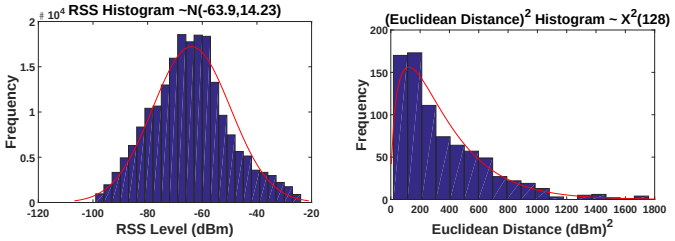
Fig. 7: Successful Predictable Channel Attack

readings that were recorded by Eve for the same packets. As shown in Figures 6(b), 6(c), and 6(d), in order to make the RSS readings for Eve similar to the RSS readings at the legitimate GO's location, $d2$ distance has to be at most 5 cm ($d2 \leq 5$ cm). For $d2 > 5$ cm, the difference in the RSS readings increases, and this makes Eve's task of obtaining the legitimate GO RSS readings more difficult.

B. Predictable channel attack defense

The RSS level in free space environments (the open air or the anechoic chamber) follows the inverse square law (the RSS level decreases 6.02 dBm each time the distance from the source is doubled) [35]. It is easy for Eve to predict the RSS level at either the client's or the legitimate GO's locations in such environments. In the real world, Eve launches her attack in public, indoor areas. In such areas, there are many static and moving objects (e.g., tables, walls, doors, passengers, customers, and students). As a result, the radio waves are reflected, diffracted, and scattered many times. Accordingly, the inverse square law is not strictly applicable, and the RSS level is very random.

In static areas, due to the lack of randomness in the wireless channel, there are few vatiations in the RSS profiles. In such areas, Eve has two approaches to obtain the same RSS profile of the client or the legitimate GO. First, is to collect many RSS profiles that simulate the client and legitimate GO communications at different physical locations. Eve uses these RSS profiles during her attack based on the physical locations of the clients and legitimate GO. However, the clients use a random transmission power each time they execute the Device Discovery and Service Discovery phases with the legitimate GO (shown in Figure 3(a). As a result, it is hard for Eve to predict the transmission power that the client uses. Second, is to pass the MDS detection by locating herself in a physical location in which her RSS values (received/overheard by all clients after they receive the acceptance for their invitations from her) equal to the legitimate GO's RSS values (received/overheard by all clients before they receive the acceptance for their invitations from Eve). This physical location has to be symmetric to the legitimate GO location corresponding to all clients in terms of distances and intermediate objects (to ensure that Eve has the same multi-path channels with all clients as the legitimate GO). In order to make the distances between Eve and any client equal to the distances between the client and



(a) (b)

Fig. 8: Google Nexus 5's histograms of the (a) RSS readings. (b) $\oplus$ between two RSS subsets.

the legitimate GO, all clients have to be in the same geometric plane/line, as shown in Figure 7. However, even if the clients exist in the same geometric plane (rare when the number of clients $\geq$ 4), the intermediate objects between Eve and all clients have to be similar to the corresponding intermediate objects between the legitimate GO and all clients, as shown in Figure 7 (also rare in public areas).

C. $\oplus$ result guessing attack defense

In order to validate whether $\oplus$ is a suitable statistical metric that can securely measure the similarity between the clients' and the legitimate GO's RSS profiles, we repeated each experiment presented in Figure 3 267 times to generate 205,056 RSS readings. First, we want to find the statistical distribution of the RSS readings. Figure 8(a) shows the histogram of the client RSS readings for the 801 experiments. The distribution of any RSS reading, say m, of the client, $rss_m \sim \mathcal{N}(\mu, \sigma^2)$. The GO RSS readings' histogram is similar to Figure 8(a) (omitted here due to space constraints). Second, we validate $\oplus$ for EvilDirectHunter.

The Euclidean distance, $d(X, Y)$, between X and Y sets of data, each of size q, is $\sqrt{\sum_{i=1}^{q}(X_i - Y_i)^2}$. At each pass of EvilDirectHunter, the client divides the $GOprfl$ into two subsets (Rss_I = $[rss_1, rss_2 \ldots rss_k]$ and Rss_{II} = $[rss_{k+1}, rss_{k+2} \ldots rss_{2k}]$). For $GOprfl$ with 256 readings, $k = 128$ for the first pass. We want to find the statistical distribution of $d^2(Rss_I, Rss_{II})$, which is the square of the Euclidean distance. Since each $rss_m \sim \mathcal{N}(\mu, \sigma^2)$, then $(rss_m - rss_r) \sim \mathcal{N}(0, 2\sigma^2)$ (m - r = k). If we assume that $Z_1, Z_2 \ldots Z_k$ are k independent normal random variables that represent $[(rss_1 - rss_{k+1}), (rss_2 - rss_{k+2}) \ldots (rss_k - rss_{2k})]$ (i.e., the square of the Euclidean distance between Rss_I and Rss_{II}), then: $d^2(Rss_I, Rss_{II}) = \sum_{i=1}^{k} Z_i^2 = \sum_{i=1}^{k}[\mathcal{N}(0, 2\sigma^2)]^2 = \sum_{i=1}^{k}[\sqrt{2}\sigma\mathcal{N}(0,1)]^2 = 2\sigma^2 \sum_{i=1}^{k} \mathcal{N}(0,1)^2$.

The sum of the squares of k independent standard normal random variables ($\sum_{i=1}^{k} \mathcal{N}(0,1)^2$) is distributed according to chi-squared distribution [37] with k degrees of freedom, $\chi^2(k)$. Due to the multiplication by $2\sigma^2$, the value of $d^2(Rss_I, Rss_{II})$ is distributed according to the scaled version of chi-squared distribution with k degrees of freedom. Figure 8(b) shows the histogram of $d^2(Rss_I, Rss_{II})$ for the 801 experiments with 128 (i.e., size of Rss_I & Rss_{II} during the first pass) degrees of freedom.

The hardness of guessing the result of $\oplus$ depends on its statistical distribution. We use Shannon entropy to measure the randomness of the statistical distributions. The hardness of guessing the result of $\oplus$ increases when the entropy of its statistical distribution increases. A valid question follows: what is the maximum entropy of $\oplus$'s statistical distribution? In probability theory and statistics, the uniform distribution is the maximum entropy distribution on any interval $[a, b]$ [38]. In order to investigate the hardness of guessing the result of $\oplus$, we compared the entropy of $\oplus$'s statistical distribution with the entropy of the uniform distribution for all passes.

For the first pass, the statistical distribution of $\oplus$ (Figure 8(b)) is distributed on the interval $[0, 1800]$ according to the scaled version of chi-squared distribution with 128 degrees of freedom, $\chi^2(128)$. The entropy of $\chi^2(k) = \frac{k}{2} + ln(2\Gamma(\frac{k}{2})) + (1 - \frac{k}{2})\psi(\frac{k}{2})$ [37]. The entropy of the uniform distribution on the interval $[0, 1800]$ is $ln(1800)$ = 7.5. We found the distributions of $\oplus$ for other passes (when $pass\#$= 2, 3, 4) in the same way we did for the first pass. Then, we compared the entropies of these distributions with the entropies of the uniform distributions on the same intervals. As shown in Figure 9, the entropy of the distribution of $\oplus$ is at least 75% of the entropy of the uniform distribution.

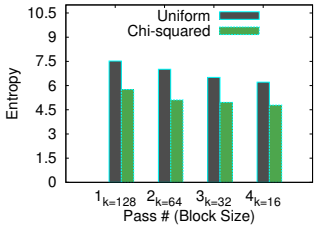


Fig. 9: Entropy of the Uniform and $\chi^2(k)$ Distributions

D. Client spoofing attack defense

EvilDirectHunter enables the legitimate GO to discover and stop Eve while she launches the client spoofing attack. As we described in Section IV-C, the client and the legitimate GO incrementally prove the mutual knowledge of the RSS profile by exchanging challenge and response packets. Since it is difficult for Eve to create an RSS profile that is similar to the client profile and to guess the result of $\oplus$ (as shown in Sections VI-A and VI-C, respectively), Eve cannot pursue EvilDirectHunter and learn the results of the $\oplus$ metric.

E. Replay attack defense

If Eve replays some/all of the client's probe-requests/GAS-requests or the legitimate GO's probe-responses/GAS-responses, the RSS profile that the client builds will be a mixture of the legitimate GO's messages and Eve's replayed messages. Indeed, the RSS profile at the client is created based on the multi-path of the channels between the client and the legitimate GO, and between the client and Eve. On the other hand, the RSS profile at Eve is created based on the multi-path of the channels between Eve and the client, and between Eve and the legitimate GO. Accordingly, the RSS profile that is created at Eve is different from the RSS profile at the client (unless Eve is very close to the client or the legitimate GO, as shown in Section VI-A). As a result, if Eve launches her attack, she will be detected by EvilDirectHunter. On the other hand, if Eve does not launch her attack and the client tries to connect to the legitimate GO, our algorithm will mistakenly claim the legitimate GO as an EvilDirect GO because the RSS profile that the legitimate GO builds will be a mixture of the client's messages and Eve's replayed messages. Essentially, if Eve causes a replay attack, she will not be able to successfully launch an EvilDirect GO attack. However,

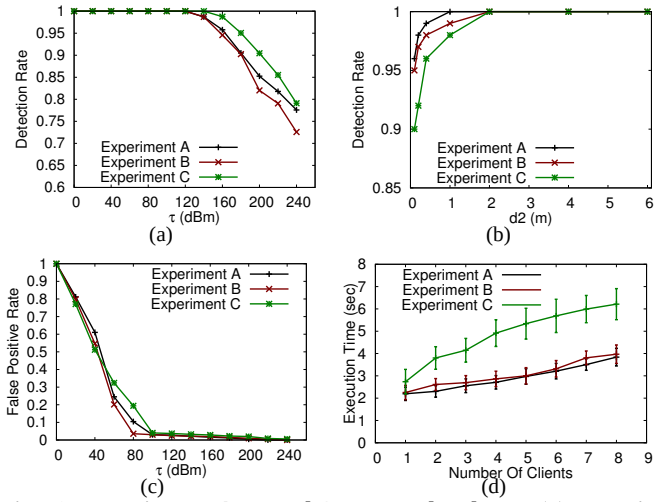


Fig. 10: Experiments A, B, and C on same hardware. (a) Detection Rate for different τ's. (b) Detection Rate for different $d2$'s. (c) False Positive Rate for different τ's. (d) Execution Time.

her replay attack will cause a denial-of-service attack (DoS) against the legitimate GO, which is not the interest of Eve.

VII. Performance Evaluation

This section presents our evaluation metrics and results.

A. Evaluation Metrics

We evaluated EvilDirectHunter using the following metrics: **1) Detection Rate**: the ability of EvilDirectHunter on detecting the attempts of the EvilDirect attacker (true positive rate), ideally 100%; **2) False Positive Rate**: the rate of claiming a legitimate GO as an EvilDirect GO, ideally 0%; **3) Execution Time (sec)**: the execution time of EvilDirectHunter.

B. EvilDirectHunter on Same Smartphones

We performed the experiments with Google Nexus 5 smartphones for the clients, the legitimate GO, and the attacker.

1) **Detection Rate**: in Section IV-D2, we tuned the values of τ when EvilDirectHunter runs on Google Nexus 5 smartphones. The value of τ impacts the detection rate of our protocol. In order to investigate the best values of τ from the detection rate perspective, we performed experiments A, B, and C with $d2$ = 2 m, and $\varepsilon = 4.53$. We varied the value of τ for the fourth pass to demonstrate its influence on the detection rate. In each experiment, the attacker used the RSS readings of the packets that he overheard while each client exchanged packets with the legitimate GO. Then, both clients executed EvilDirectHunter with the attacker's smartphone. We counted the number of times that EvilDirectHunter was able to detect the attacker as EvilDirect GO for both clients. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment.

Figure 10(a) shows the detection rate of EvilDirectHunter for both clients for experiments A, B, and C. As shown in this figure, the detection rate of EvilDirectHunter decreases when τ increases. Indeed, the values of τ that we tuned from the Monte Carlo simulation for Google Nexus 5 smartphones in Table II achieve a 100% detection rate for all experiments. The additional phase for static environments (Section IV-C) executed for all runs of experiment C.

Moreover, we investigated the effect of $d2$ distance on the detection rate. We performed experiments A, B, and C while we varied $d2$ distance between 10 cm to 6 m to investigate its effect on the detection rate. For each experiment, we repeated the same steps above. However, we used the τ values for the four passes presented in the second column of Table II. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment. Figure 10(b) shows the detection rate of EvilDirectHunter for both clients for different $d2$ values. As is clear in this figure, the detection rate is 100% for A, B, and C experiments when $d2 > 1$ m.

2) **False Positive Rate**: in order to investigate the best values of τ from the false positive rate perspective, we performed experiments A, B, and C with $\varepsilon = 4.53$. We varied the value of τ for the fourth pass to demonstrate its influence on the false positive rate. In each experiment, both clients executed EvilDirectHunter with the legitimate GO. We counted the number of times that EvilDirectHunter mistakenly detected the legitimate GO as EvilDirect GO for both clients. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment. Figure 10(c) shows the false positive rate when we vary the value of τ for the fourth pass. As is clear in this figure, the false positive rate decreases when τ increases. Our goal is to achieve a 0% false positive rate for our protocol. Even though increasing τ values achieves a very low false positive rate, we are unable to use high τ values because they will lower the detection rate as is presented in Figure 10(a). Thus, there is a tradeoff between the detection rate and the false positive rate. We found that the highest false positive rate for all experiments using the τ values for Google Nexus 5 smartphones in Table II is $\sim$ 4.0%.

3) **Execution Time**: in order to evaluate the execution time of EvilDirectHunter, we repeated experiments A, B, and C while we increased the number of clients between one and eight devices. These clients simultaneously run EvilDirectHunter with the legitimate GO. We measured the execution time needed to run EvilDirectHunter on all clients. We repeated each experiment 40 times, and we averaged these 40 runs for each experiment. Figure 10(d) shows the execution time for all experiments, with error bars showing standard deviation. As is clear in this figure, by increasing the number of clients, the execution time increases linearly. Accordingly, the legitimate GO is able to run EvilDirectHunter, and reply to all clients' challenges in a short time. Due to the additional phase for static environments, the execution time for experiment C is higher than the execution times for experiments A and B.

C. EvilDirectHunter on Different Smartphones

We investigated the robustness of EvilDirectHunter on smartphones with different hardware components by repeating the same steps of all experiments presented in Section VII-B. However, we used Samsung Galaxy S2 smartphones for the two clients, and Google Nexus 5 smartphones for both the legitimate GO and the attacker. Figure 11(a) shows that the detection rate of EvilDirectHunter for both clients decreases when the τ value for the fourth pass increases. The values of τ that we calculated from the Monte Carlo simulation for Google

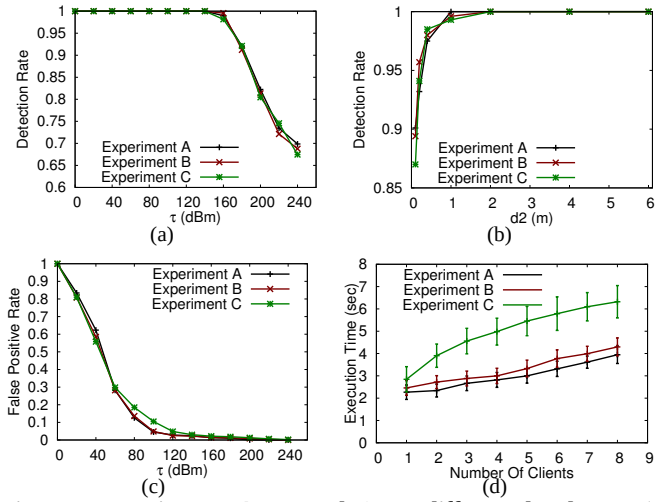


Fig. 11: Experiments A, B, and C on different hardware. (a) Detection Rate for different τ's. (b) Detection Rate for different $d2$'s. (c) False Positive Rate for different τ's. (d) Execution Time.

Nexus 5 and Samsung Galaxy S2 smartphones (third column of Table 2) achieve a 100% detection rate for experiments A, B, and C. As shown in Figure 11(b), the detection rate is 100% for A, B, and C experiments when $d2 > 1$ m. Figure 11(c) shows the false positive rate when we vary the value of τ for the fourth pass. The highest false positive rate for all experiments using the τ values for Google Nexus 5 and Samsung Galaxy S2 smartphones (third column of Table 2) is $\sim$ 4.5%. The execution time (shown in Figure 11(d)) of EvilDirectHunter on different smartphones is similar to the execution time on similar smartphones (i.e., Figure 10(d)).

VIII. Conclusions and Future Work

We presented a novel protocol, EvilDirectHunter, for detecting EvilDirect attacks in Wi-Fi Direct for both static and dynamic environments. EvilDirectHunter exploits the randomness in the wireless channel between the clients and the GO as the source for detecting the EvilDirect attacks. Our evaluations show that EvilDirectHunter is able to identify EvilDirect GOs with a very high detection rate while maintaining a very low false positive rate. In our future work, we plan to validate EvilDirectHunter detection capabilities in other environments with many heterogeneous hardware components. We also plan to study the values of ε and τ parameters for these environments and hardware components. Moreover, we plan to improve EvilDirectHunter to be able to defend against other types of man-in-the-middle attacks.

Acknowledgment

We thank Mustafa Alshawaqfeh and Chen Yang for helpful discussions on an early version of this paper. This work was funded by NSF grant #1127449, #1145858, and by the German Jordanian University.

References

[1] D. Camps-Mur, A. G. Saavedra, and P. Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *IEEE Wireless Communications Magazine*, 20(3), 2013.
[2] Abi research, https://www.abiresearch.com/.
[3] Wi-fi protected setup specification version 1.0h. 2006.
[4] S. Viehbck. Wi-fi protected setup online pin brute force vulnerability, cert vulnerability note vu#723755. 2011.
[5] D. Bongard. Wi-fi protected setup pixie dust offline pin brute force vulnerability. 2014.
[6] A. Altaweel, R. Stoleru, and S. Mandal. On secure shared key establishment for mobile devices using contextual information. *IPCCC'15*.
[7] S. Mandal, C. Yang, A. Altaweel, and R. Stoleru. An efficient pairwise key establishment scheme for ad-hoc mobile clouds. *WiMob'15*.
[8] W. Wei, S. Jaiswal, J. Kurose, D. Towsley, K. Suh, and B. Wang. Identifying 802.11 traffic from passive measurements using iterative bayesian inference. *IEEE Transactions on Networking*, 20(2), 2012.
[9] W. Wei, K. Suh, B. Wang, Y. Gu, J. Kurose, D. Towsley, and S. Jaiswal. Passive online detection of 802.11 traffic using sequential hypothesis testing with tcp ack-pairs. *IEEE Transactions on Mobile Computing*, 8(3), 2009.
[10] A. Venkataraman and R. Beyah. Rogue access point detection using innate characteristics of the 802.11 mac. *SecureComm'09*.
[11] S. Jana and S. K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *MobiCom'08*.
[12] Tired of rogues? solutions for detecting and eliminating rogue wireless networks, white paper, https://www.zebra.com/. 2011.
[13] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. Enhancing the security of corporate wi-fi networks using dair. *MobiSys'06*.
[14] H. Mustafa and W. Xu. Cetad: Detecting evil twin access point attacks in wireless hotspots. *CNS'14*.
[15] C. Yang, Y. Song, and G. Gu. Active user-side evil twin access point detection using statistical techniques. *IEEE Transactions on Information Forensics and Security*, 7(5), 2012.
[16] H. Alipour, Y. B. Al-Nashif, P. Satam, and S. Hariri. Wireless anomaly detection based on ieee 802.11 behavior analysis. *IEEE Transactions on Information Forensics and Security*, 10(10), 2015.
[17] R. Gill, J. Smith, and A. Clark. Experiences in passively detecting session hijacking attacks in ieee 802.11 networks. *ACSW'06*.
[18] X. Long and B. Sikdar. A mechanism for detecting session hijacks in wireless networks. *IEEE Transactions on Wireless Communications*, 9(4), 2010.
[19] S. Mathur, W. Trappe, N. Mandayam, C. Ye, and A. Reznik. Radio-telepathy: Extracting a secret key from an unauthenticated wireless channel. *MobiCom'08*.
[20] I. Borg and P. J. F. Groenen. Modern multidimensional scaling, series in statistics, springer. 2005.
[21] W. K. Edwards. Discovery systems in ubiquitous computing. *IEEE Pervasive Computing*, 5(2), 2006.
[22] 802.11u-2011, wireless lan medium access control (mac) and physical layer (phy) specifications amendment 9: Interworking with external networks. 2011.
[23] Google play store, https://play.google.com/store.
[24] W. Wei, B. Wang, C. Zhang, J. Kurose, and D. Towsley. Classification of access network types: Ethernet wireless lan, adsl, cable modem or dialup? *INFOCOM'05*.
[25] H. Han, B. Sheng, C.Tan, Q. Li, and S. Lu. A measurement based rogue ap detection scheme. *INFOCOM'09*.
[26] V. Baiamonte, K. Papagiannaki, and G. Iannaccone. Detecting 802.11 wireless hosts from remote passive observations. *NETWORKING'07*.
[27] S. Shetty, M. Song, and L. Ma. Rogue access point detection by analyzing network traffic characteristics. *MILCOM'07*.
[28] R. Beyah, S. Kangude, G. Yu, B. Strickland, and J. Copeland. Rogue access point detection using temporal traffic characteristics. *GLOBECOM'04*.
[29] H. Yin, G. Chen, and J. Wang. Detecting protected layer-3 rogue aps. *BROADNETS'07*.
[30] L. Watkins, R. Beyah, and C. Corbett. A passive approach to rogue access point detection. *GLOBECOM'07*.
[31] Wisentry, http://www.wimetrics.com/.
[32] The airmagnet project, http://www.airmagnet.com/.
[33] Rogue access point detection: Automatically detect and manage wireless threats to your network, white paper, http://www.proxim.com/. 2004.
[34] Radu Stoleru, Haijie Wu, and Harsha Chenji. Secure neighbor discovery in mobile ad hoc networks. *MASS'11*.
[35] W. Debus. Rf path loss and transmission distance calculations, axonn, technical memorandum. 2006.
[36] Cyanogenmod code, http://wiki.cyanogenmod.org/.
[37] National Institute of Standards and Technology. Engineering statistics handbook. 2003.
[38] C. Marsh. Introduction to continuous entropy. 2013.