A Novel Traceroute-based Detection Scheme for Wi-Fi Evil Twin Attacks

Alex Burns[1], Longfei Wu[1], Xiaojiang Du[1], Liehuang Zhu[2]
[1]Department of Computer and Information Science, Temple University, Philadelphia, PA, USA, 19122
[2]School of Computer Science, Beijing Institute of Technology, Beijing, China, 100081
Email: {tuf42726, longfei.wu, dux}@temple.edu, liehuangz@bit.edu.cn

***Abstract*—Wireless sensor networks (WSNs) are widely deployed to collect data from surrounding context. Then, a sink node will aggregate the sensing data and send the data to the remote server or user via Internet. The most popular wireless Internet access technology used in WSNs is Wi-Fi. However, the data may be leaked if the access through Wi-Fi is not well-guarded. Wi-Fi hotspots are deployed in an unprecedented speed to facilitate people's lives. The open access nature makes them vulnerable to an evil twin access point (AP), which has the same service set id (SSID) as the legitimate AP and larger signal strength. Current Wi-Fi capable devices (e.g., the sink node in WSNs) are not able to detect the evil twin attack, and will automatically switch to the bogus AP. In this paper, we devise a novel detection scheme based on the commonly used network diagnostic tool traceroute. A remote detection server is set up so that the client-to-server and server-to-client traceroute results are compared. If the evil twin AP is present, it will attempt to conceal the legitimate AP. The inconsistency among the two traceroute results will reveal the evil twin attack. We first present the attack model, then describe the detection scheme in detail. In our implementation, a Nexus 4 smartphone serves as the client, a desktop PC with a USB wireless adapter is set up as the evil twin AP, and the detection service is running on an Amazon EC2 Server. The experimental result demonstrates that our scheme can effectively detect an evil twin attack.**

***Index Terms*—Wi-Fi security; evil twin attack; traceroute**

I. Introduction

Wireless sensor networks (WSNs) are being increasingly popular and have become a key enabler for the Internet of Things (IoT) applications. WSNs play an important role in collecting data from the physical world. The data collected by the sensor nodes are all forwarded to a more powerful sink node or terminal IoT device, which can connect to the Internet with various access technologies. The most common Internet access technology used today is Wi-Fi. The Wi-Fi Alliance reported that the shipments of Wi-Fi devices are expected to surpass 15 billion by the end of 2016 [1], demonstrating the vast reach of Wi-Fi capable devices.

Wi-Fi compliant devices connect to the wireless local area network (WLAN) through a Wi-Fi AP. Many open APs (also known as hotspots) are deployed in public places, such as airports, restaurants, hotels, to provide free Internet service to their customers. However, such Wi-Fi hotspots are vulnerable to evil twin attacks, in which a rogue Wi-Fi access point that appears to be a legitimate one offered on the premises, but actually has been set up to eavesdrop on wireless communications. Particularly, the evil twin AP has the potential to be performing various security attacks on the user such as various Man in the Middle (MITM) attacks. Encrypted information can be harvested using techniques such as SSL Stripping [2].

The detection of evil twin attacks has recently received enormous attention from many researchers. Using a VPN is one way to safeguard against compromised network nodes but is not always an appropriate solution, neither is relying on network administrators to secure the network. A client based detection scheme is needed to verify that a hotspot is not malicious before the user sends sensitive data over the connection. Unfortunately it is not a simple task to identify a malicious access point from a benign access point if not managing the network. To the users, the evil twin AP provides a connection to the Internet and appears to provide the same service set id (SSID) as a legitimate AP.

In this paper, we analyze the feature of evil twin attacks, and propose a bi-directional traceroute-based detection scheme that utilizes a remote detector server. Specifically, the hops in the local area network on both the client-to-server path and the server-to-client path are compared. Our scheme can make a deterministic and accurate decision on whether the client device is connecting to an evil twin AP. We implement and evaluate the proposed scheme with real experiments.

The rest of the paper is organized as follows. Section 2 discuss the related work. Section 3 introduces the attack model and the traceroute. Section 4 describes our detection scheme. The implementation and experimental evaluation are presented in section 5 and section 6, respectively. Finally, we present the conclusion for this work in section 7.

II. Related Work

There are generally two types of solutions to detect evil twin attacks.

The first type of solutions is based on the fingerprints of a predefined authorized AP list [3]–[5]. However, this type of solutions are usually conducted by the network administrator or require the collaboration of the administrator, which is not available for most of Wi-Fi networks.

The second type of solutions are client-side detection schemes. One group of works are based on the packet transmission delay [6]–[8]. The commonly used measurements include Inter-packet Arrival Time (IAT) and Round Trip Time (RTT). The limitation of RTT/IAT-based detection methods is that the transmission delay may be due to a variety of reasons, e.g.,

interferences, collisions, network topology changes, etc, it is not accurate to consider the number of hops as the only cause of the delay. Other client-side detection schemes include [9]–[12].

In [9], rogue access points are differentiated from evil twin access points, which is important because they assert that their solution uniquely detects both MITM rogue APs and evil twin APs. Their detection scheme where they compare two APs is discussed. If different IPs and different network IDs are detected, traceroute is ran to the same destination. If there is an extra hop in the result, which is proof of man in the middle, the user is alerted. If the traceroute indicates no extra hop, and just indicates different routes to the same destination, then their detection scheme cannot determine which AP is the rogue AP, and simply warns the user that the network is unsafe. However, there is no guarantee that the two traceroute paths are completely the same (e.g., network topology changes).

In [10], a client-side evil twin AP detection technique is proposed. They demonstrate that when a user is connected to an evil twin AP the packets travel two wireless hops, one between the user and the evil twin AP and one between the evil twin AP and the legitimate AP. This provides a time delay over the single hop that connecting to a legitimate access point provides. Their proposed detection scheme exploits this timing difference to detect the presence of an evil twin AP. They had issues with their algorithm performance when wireless traffic was saturated, with a 65% detection accuracy. They also discuss that a traceroute based detection scheme may be evaded through evil twin behavior, where the SSID and MAC parameters are copied in an attempt to fool the user into interpreting the traceroute results incorrectly.

In [11], spoofing is explained. Crucially, in their explanation of spoofing an upstream connection to the Internet is not provided. In their description of a man in the middle attack, an upstream connection is added via Wi-Fi or a 3G or similar cellular based interface. The attacker can eavesdrop unless the user is protected by end-to-end encryption such as with a VPN or SSL. Even with SSL, the user may accept invalid SSL certificates and be exposed. Their protocols used to check the authenticity of an AP is to use two types of evidence, physical linkage and virtual linkage. Their paper does not feature an algorithm to detect rogue APs, instead focusing on users rationales when comparing various network authentication methods. It is also cumbersome and complex in that it involves a large public display linking using physically linked tokens.

In [12], a client-side detection method using SSL/TCP connection to a remote web server is used to attempt to detect the changing of the gateways public Internet Protocol (IP) address by switching from one AP to another in the middle of the SSL/TCP connection. This involves connecting to a remote random web server such as *www.google.com*, using a 3-way TCP handshake. Once complete, the client switches to a new AP and sends a GET HTML request to download a webpage from the remote server. If the APs use the same gateway, the TCP connection will not break, otherwise, the TCP connection will break which indicates the APs use different gateways in an attempt to identify evil twin APs.

In [13], the detection is based on the packet forwarding behavior, which is a key property of evil twins. Evil Twin (ET) detector is used to determine whether an AP forwards wireless packets to another AP. Specifically, ET detector changes the wireless network interface controller (WNIC) of a laptop to monitor mode, so that the laptop can capture all packets that are transmitted through the channel monitored by the WNIC. However, it is not practical to deploy an extra ET detector device along with each Wi-Fi AP.

III. PROBLEM STATEMENT

Wi-Fi capable devices are widespread and growing rapidly. Users of these devices connect to wireless APs in various environments including at work and in public locations without considering if the AP is safe to connect to. Currently, client devices do not have native methods or widely available programs to determine if an access point exhibits malicious or suspicious behavior and should be avoided. Evil twin APs are malicious access points which may deploy various schemes to capture sensitive traffic such as passwords and disguise their presence, such as identity spoofing. Various methods exist to detect evil twin APs but typically depend on network administration abilities, round trip timing measurements, or extra monitoring device which have limitations. Instead, we propose a solution using bi-directional traceroutes.

A. Background

The Wi-Fi APs broadcast their SSIDs as the identification, and the client devices can choose the network they own or have subscribed to associate with. The range of a Wi-Fi AP is limited, so large enterprise or other organization may deploy multiple APs to improve the coverage. These APs can have the same SSID, the client device will automatically associate with the one that has the highest Received Signal Strength Indication (RSSI). Figure 1 depicts the general architecture of normal Wi-Fi hotspots.

B. Attack Model

There are generally two types of evil twin attacks based on how the evil twin AP connects to the Internet. The first type is the ① cellular-based evil twin attacks which use cellular Internet access, e.g., 2G/3G/4G, to provide Internet access to the client devices. The second type is the ② relay-based evil twin attacks which utilize the legitimate AP's Internet access, by relaying the traffic through the legitimate AP. In this attack, the evil twin AP needs to associate with the legitimate AP as a client, and creates its own Wi-Fi hotspot to serve the victim client devices. For both kind of attacks, the evil twin AP has to attract the victim client to disconnect from the legitimate AP and connect to itself. Since the Wi-Fi capable devices automatically connect to the AP with the maximum RSSI value, the victim client will associate with the evil twin AP if it has the same SSID and its RSSI is higher than that of the legitimate AP. In this work, we mainly focus on the relay-based evil twin attacks (illustrated in Figure 2).

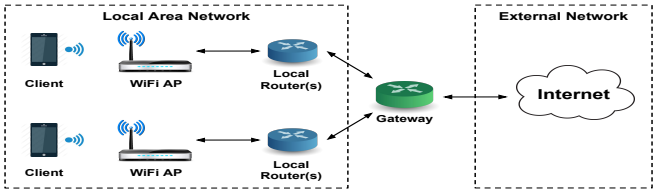


Fig. 1. Normal Wi-Fi Hotspots Architecture

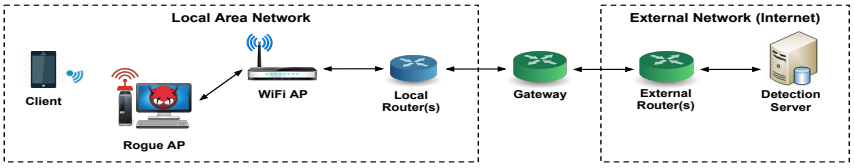


Fig. 2. Relay-based Evil Twin Attacks

C. Traceroute

Traceroute is a computer network diagnostic tool for tracking the route (path) and measuring transit delays of packets across an IP network [14]. It utilizes the time-to-live (TTL) field to determine the intermediate routers being traversed towards the destination. The initiator sends traceroute packets with gradually increasing TTL value, starting with a TTL value of one. The receiver routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP_Time_Exceeded. Proceeding in this way, traceroute builds a list of routers that packets traverse based on the returned ICMP_Time_Exceeded messages, until the destination is reached and returns an ICMP Echo Reply message.

An important note is that traceroute is asymmetric, and only shows the forward path. The reverse path taken by the return can be completely different from every hop in the forward path, due to the asymmetry of the routing tables in neighboring routers or the topology changes. This can affect the latency and is a reason why using latency information from traceroute is an inadequate method to identify evil twin access points. This also demonstrates that simply comparing the traceroute from the source to the destination, then from the destination to the source is not enough to detect the evil twin AP. Instead of comparing the entire path, we examine only the hops in the subnetwork (local network).

In the local area network, the Network Address Translation (NAT) technology [15] is used to map the client device's private IP address to the specific port of the gateway. The client-to-gateway and the gateway-to-client traceroutes are symmetric if there is no violation of the protocol or manipulation of traceroute packets.

IV. Our Detection Scheme

To protect the client from switching to the evil twin AP, the detection has to be performed each time the client disconnects from the current Wi-Fi AP and reconnects to another with the same SSID. In normal cases, multiple legitimate APs may be deployed to cover a large area. The proposed scheme should be able to differentiate the switching to an evil twin AP and to another legitimate AP. As shown in Figure 2, it takes one more hop to the destination when switching to an evil twin AP than to a legitimate AP. Intuitively, to avoid of being exposed, the evil twin AP has to hide the legitimate AP from the client. To make the legitimate AP "disappear", the evil twin must tamper those traceroute packets coming from the client and going towards the hops after it. Without modifying its system's default traceroute program, the evil twin can just increase the TTL of all such traceroute packets by one, so that the legitimate AP is always "jumped over" while other hops behind appear in the same manner as before in the traceroute results that the client get. Hence, the client-issued traceroute packets alone are insufficient to detect the evil twin attack.

We propose to use a dedicated detection server to assist the detection, which also conducts traceroute in the reverse direction. Although the evil twin AP can tamper the client-to-server traceroute result to hide the existence of the legitimate AP, it cannot prevent the detection server discovering the legitimate AP from the server-to-client traceroute result. This is because it is located behind the legitimate AP from the server's perspective, hence is not able to intercept the ICMP_Time_Exceeded message of the legitimate AP or the gateway returned to the server. As mentioned earlier, we use the hops within the local network for the detection, hence only the local traceroute result from the client to the gateway is encrypted and transferred to the server for comparison. We first need to make sure that the new AP connects to the Internet through the same gateway as before the switching. That is to say, the new AP is within the same subnetwork as the old one. Next we check if the two local traceroute results are inconsistent, especially if any hop has been intentionally "erased" from the client-to-server traceroute in the subnetwork. If so, a typical evil twin attack is detected; else if any other inconsistency is found, the client will also be alerted; otherwise, the client will receive a "safe" message instead.

In our scheme, we assume that the owners of the client devices have already subscribed the evil twin detection service, so that the communications between the client and the server

are encrypted and cannot be tampered with. Besides, the intentional drop of the traceroute packets or other messages between the client and the server is beyond the scope of this paper, such abnormality will be enough to alert the user of potential attacks (e.g., denial-of-service attack).

V. Implementation

This section introduces the experimental setup and implementation. The detection server is an Amazon EC2 t2 micro instance with 1 vCPU and 1 GB of memory, which is sufficient to run lightweight tasks like traceroute to a single client. If this solution was scaled up to serve a large number of clients, a more powerful server is required. The client device is a Nexus 4 Android phone running Cyanogenmod 12.1. We use a router and modem combination unit provided by Comcast to implement the legitimate AP. The evil twin is a Windows PC connected to the legitimate AP, and the Alfa AWUS036NHA USB wireless adapter is plugged in as the evil twin AP.

A. Traceroute from client to server

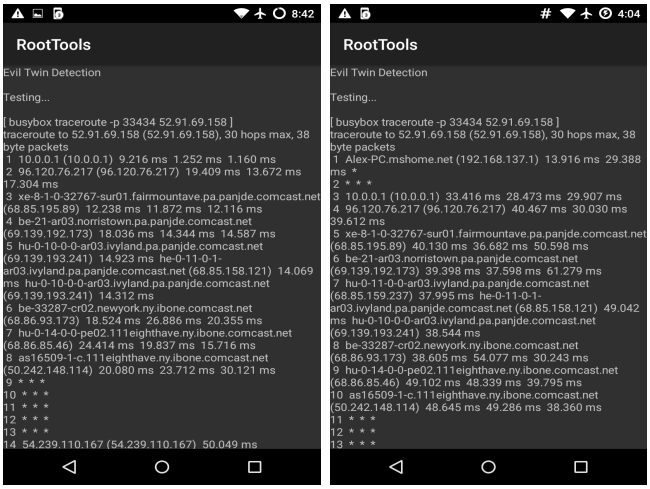
Traceroute on an Android phone is not nearly as simple as on a typical desktop PC or laptop. In Windows or Linux, traceroute or tracert is already installed and the user simply has to open a terminal and input the command. Such functions are not available to the user on an Android phone. We use Cyanogenmod which has the Busybox binaries built in [16]. Busybox contains lightweight versions of common UNIX utilities into one small executable, including the traceroute and the Netcat (introduced below). The detection app running on the client phone can programmatically invoke these commands using a Java library named RootShell [17].

B. Traceroute from Server to Client

Since the client is in the local network and only has a private IP, the gateway will assign a specific port for the client. The detection server can learn this port by looking up the established TCP connections with "netstat" command, and selecting the one with the target user's IP address and program name. The server-to-client traceroute packets are then sent to this port for address translation, and then forwarded to the client. However, the assigned port will be revoked when the TCP connection between the client and the server ends (e.g., after the client-to-server traceroute finishes), so that the server-to-client traceroute sent to that port cannot proceed past the gateway. This issue can be solved by Netcat, a UNIX utility which reads and writes data across network connections using TCP or UDP. The client is able to maintain a TCP connection to the server with Netcat, until the server has returned the detection result and the client explicitly disconnects from the server. On the server end, we use an easy-to-use program InTrace [18] to perform traceroute. which enables users to enumerate IP hops using existing TCP connections, both initiated from local network or from remote hosts.

VI. Performance Evaluation

In this section, we conduct experiments to evaluate the performance of our scheme.



(a) Traceroute without evil twin AP (b) Traceroute with evil twin AP

Fig. 3. Client-to-server Traceroute Result

Figure 3 displays the client-to-server traceroute result with and without the evil twin AP. When no evil twin AP is involved (Figure 3(a)), the first hop "10.0.0.1" is the original legitimate AP. When the client switches to the evil twin AP (Figure 3(b)), the first hop "Alex-PC.mshome.net (192.168.137.1)" is the evil twin AP and the second hop "10.0.0.1" is the legitimate AP. The evil twin can manipulate the traceroute packets so that it appears to be the legitimate AP with the IP address "10.0.0.1" and the original legitimate AP disappears from the client-to-server traceroute. After the manipulation, the fake client-to-server traceroute looks exactly the same as Figure 3(a).

The detection procedure on the server is presented as below:

1) Create two files, one for the client to server traceroute results called "nc_out" and one for the server to client traceroute results called "intrace_out".
2) Create two named pipes, this is for directing input and output from programs running in the background.
3) Start Netcat, run continuously in background using the k and l flags and listen to port 44544. Send the input through a named pipe and the output to "intrace_out".
4) Wait for a TCP connection to be established. This can be done in various ways. For example, use Netstat to continuously view the contents and Grep for changes. Then poll for changes in the output file. Once Netcat was outputting, it means a TCP connection was established.
5) Get the Netcat foreign IP and port. To do this we used awk to parse the Netstat results. To save the result to a variable, the following code was used:

```
NCIP="$(sudo netstat -anpt|awk 'BEGIN {FS="[ :]+"};
/ESTABLISHED/ && /nc/{print $6}')"
```

   As shown in Figure 4, under the ESTABLISHED TCP connections containing NC, field 6 is the foreign IP and field 7 is the foreign port. This IP is also used to judge if the client is connecting through the same gateway as before the switching.
6) Once the TCP connection has been successfully made and data has been sent across, InTrace executes using

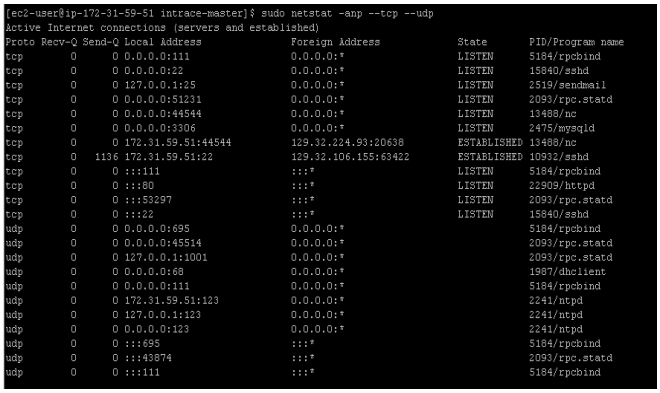


Fig. 4. Netstat result

the extracted foreign IP and port. The input is redirected to come from the second named pipe, and the output is directed to "intrace_out". Figure 5 shows the InTrace running on the EC2 server, the destination is the client connected to the legitimate AP. In the ouput, the hop 15, 16, and 17 are the legitimate AP, evil twin AP and the client, respectively.

7) Once InTrace successfully packet sniffs the established TCP connection to the client, the output results change to include the string "ENTER". We echo the enter keystroke and sent it to the second named pipe. InTrace now begins the traceroute to the client.
8) We constantly poll "intrace_out" in reverse and grep for "[TCP]". Once found the client has been reached and InTrace should quit, the process is killed using "pkill".
9) Now we have the client to server results in "nc_out" and the server to client results in "intrace_out". As established, the important information is the local subnet.
10) "nc_out" is parsed using regular expressions to determine how many hops are in the local IP space. This determines the number of hops before leaving the local subnet.
11) "intrace_out" is parsed to determine how many results after the source address equals the destination address exist before the final result of "[TCP]" which indicates the client was reached.
12) These are compared and this result is returned to the client via the active Netcat connection.

As we can see, the manipulated client-to-server traceroute result (Figure 3(a)) only contains the fake legitimate AP in the subnetwork, while the server-to-client traceroute result contains two "ICMP_TIMXCEED NAT" hops. The inconsistency indicates that the client is under an evil twin attack.

VII. Conclusion

In this work, we proposed a novel traceroute-based detection scheme for evil twin attacks. We implemented the scheme with real devices. The experiment shows that it is effective in detecting relay-based evil twin APs. In our future work, we will continue to work on the detection for the cellular-based evil twin attacks.

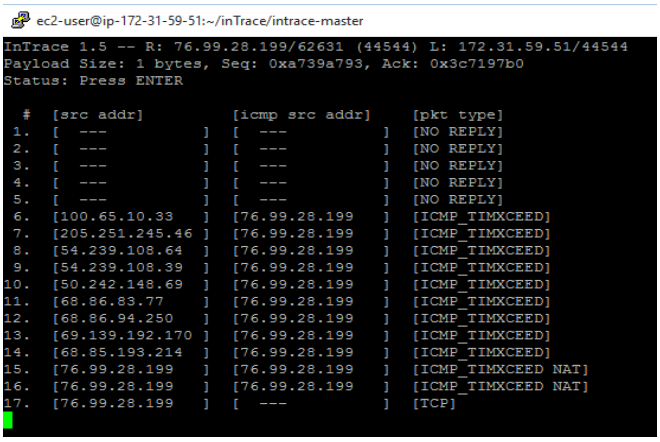


Fig. 5. InTrace result

Acknowledgment

This work was supported in part by the US NSF under grants CNS-1022552 and CNS-1564128.

References

[1] Wi-Fi Alliance Press Release, "Wi-fi device shipments to surpass 15 billion by end of 2016," http://www.wi-fi.org/news-events/newsroom/wi-fi-device-shipments-to-surpass-15-billion-by-end-of-2016, 2016.
[2] K. Seifried, "Attacks against ssl," *Linux Pro Magazine*, March, 2010.
[3] H. Gonzales, K. Bauer, J. Lindqvist, D. McCoy, and D. Sicker, "Practical defenses for evil twin attacks in 802.11," in *IEEE Global Telecommunications Conference (GLOBECOM)*, 2010.
[4] B. Alotaibi and K. Elleithy, "An empirical fingerprint framework to detect rogue access points," in *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2015.
[5] K. Elleithy and B. Alotaibi, "A passive fingerprint technique to detect fake access points," in *IEEE Wireless Telecommunications Symposium*, 2015.
[6] H. Mustafa and W. Xu, "Cetad: Detecting evil twin access point attacks in wireless hotspots," in *IEEE Conference on Communications and Network Security (CNS)*, 2014.
[7] Y. Song, C. Yang, and G. Gu, "Who is peeping at your passwords at starbucks? to catch an evil twin access point," in *IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, 2010.
[8] I. Kim, J. Seo, T. Shon, and J. Moon, "A novel approach to detection of mobile rogue access points," *Security & Communication Networks*, 2013.
[9] S. Nikbakhsh, A. B. A. Manaf, M. Zamani, and M. Janbeglou, "A novel approach for rogue access point detection on the client-side," *27th International Conference on Advanced Information Networking and Applications Workshops*, 2012.
[10] H. Han, B. Sheng, C. C. Tan, Q. Li, and S. Lu, "A measurement based rogue ap detection scheme," in *IEEE INFOCOM*, April 2009.
[11] T. Kindberg, C. Bevan, E. O'Neill, J. Mitchell, J. Grimmett, and D. Woodgate, "Authenticating ubiquitous services: A study of wireless hotspot access," in *11th International Conference on Ubiquitous Computing*, 2009.
[12] O. Nakhila, E. Dondyk, M. F. Amjad, and C. Zou, "User-side wi-fi evil twin attack detection using ssl/tcp protocols," in *12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015.
[13] F.-H. Hsu, C.-S. Wang, Y.-L. Hsu, Y.-P. Cheng, and Y.-H. Hsneh, "A client-side detection mechanism for evil twins," *Computers & Electrical Engineering*, 2015.
[14] Traceroute, https://en.wikipedia.org/wiki/Traceroute.
[15] Network Address Translation (NAT), https://en.wikipedia.org/wiki/ Network_address_translation).
[16] Traceroute Android_External_Busybox, https://github.com/ Cyanogen-Mod/android_external_busybox.
[17] S. Ericson, "Rootshell v 1.3," https://github.com/Stericson/ RootShell, 2015.
[18] R. Swiecki, "Intrace," https://github.com/robertswiecki/intrace, 2007.