FakeAP Detector: An Android-Based Client-Side Application for Detecting Wi-Fi Hotspot Spoofing

Article *in* IEEE Access · January 2022

DOI: 10.1109/ACCESS.2022.3146802

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 3, 2022, accepted January 21, 2022, date of publication January 27, 2022, date of current version February 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3146802

FakeAP Detector: An Android-Based Client-Side Application for Detecting Wi-Fi Hotspot Spoofing

LUNODZO J. MWINUKA[1], ABEL Z. AGGHEY[2], SHUBI F. KAIJAGE[1], (Senior Member, IEEE), AND JEMA D. NDIBWILE[3]

[1]School of Computation and Communication Science and Engineering, The Nelson Mandela African Institution of Science and Technology, Arusha 44740, Tanzania
[2]Information Security and Communication Technology, Kamili Technologies Ltd., Dar es Salaam 20640, Tanzania
[3]College of Engineering, Carnegie Mellon University Africa, Kigali, Rwanda

Corresponding author: Lunodzo J. Mwinuka (mwinukal@nm-aist.ac.tz)

ABSTRACT Network spoofing is becoming a common attack in wireless networks. The trend is going high due to an increase in Internet users. Similarly, there is a rapid growth of numbers in mobile devices in the working environments and on most official occasions. The trends pose a huge threat to users since they become the prime target of attackers. More unfortunately, mobile devices have weak security measures due to their limited computational powers. Current approaches to detect spoofing attacks focus on personal computers and rely on the network hosts' capacity, leaving guest users with mobile devices at risk. Some approaches on Android-based devices demand root privilege, which is highly discouraged. This paper presents an Android-based client-side solution to detect the presence of fake access points in a perimeter using details collected from probe responses. Our approach considers the difference in security information and signal level of an access point (AP). We present the detection in three networks, (i) open networks, (ii) closed networks and (iii) networks with captive portals. As a departure from existing works, our solution does not require root access for detection, and it is developed for portability and better performance. Experimental results show that our approach can detect fake access points with an accuracy of 99% and 99.7% at an average of 24.64 and 7.78 milliseconds in open and closed networks, respectively.

INDEX TERMS Android, client-side, intrusion detection, evil twin, network spoofing, rogue access point.

I. INTRODUCTION AND BACKGROUND

The global mobile population on the Internet is rapidly expanding, accounting for 48% of global online page views and 50.44% of Internet traffic [1], [2]. Mobile devices have also become a preferred choice for usage in education [3], [4], communication, and e-commerce [5] due to the sensitivity of the information they store [6], as well as their ease of use, portability, and reliable functions. On the other hand, most mobile phone users opt for wireless networks as their means to access the Internet. As a result, the use of wireless networks is also exponentially growing [7]. Internet services, voice over wireless, health care, education, and agriculture services all make use of it. Security risks occur as wireless communication becomes incorporated into important commercial operations involving financial and personal data as it becomes part of regular digital routines, including Internet access. Furthermore, wireless routers actively broadcast the unencrypted beacon packets to associate a client access point (AP), making the situation even more worrisome [8]. Thus, wireless security has become more vital to Internet users as it becomes the backbone of the Internet connection.

Because of the openness nature of wireless networks, users are vulnerable to wireless attacks in which attackers can readily access personal and financial information, which could then be used to carry out various types of attacks [9]. One possible attack on wireless networks is the spoofing attack, sometimes referred to as Evil Twin Attack (ETA), KARMA, rogue access points (RAP) or network spoofing attack. Spoofing attacks on the Internet work in an environment where information is transmitted between network users who are identified by Internet addresses. The sender or receiver address of a successful spoofing attack is disguised to appear legitimate. As a result, the receiver does not notice the sender's true address, and the sender sends packets to a bogus or spoofed address [10]. An attack occurs when an attacker successfully creates illegitimate Wi-Fi APs in wireless networks, and a user connects to it. Rogue Wi-Fi

The associate editor coordinating the review of this manuscript and approving it for publication was Stefano Scanzio.

hotspots are one of the simplest ways for attacking users in organisations, Internet cafés, universities, airports, and other public areas. Hence this type of assault is considered risky. Despite its simplicity, it has far-reaching consequences for users that are as bad as any other spoofing assault [11]. The report by Kaspersky Lab [12] mentions network spoofing as one of the major mobile security threats since they give room for many other forms of attacks in a network [13]–[15].

In the Wi-Fi hotspot spoofing attack, an attacker creates an open hotspot with a name similar to the host organisation or common public Wi-Fi or sometimes assigns deceiving names such as FastWiFi, OpenAccess5G etc. Alternatively, attackers may deauthenticate a user from AP and suppress the original AP signals while boosting theirs with a duplicate AP name. This is done to allow users to attempt reauthentication with the RAP. In other forms, ETA could be created to mimic the Service Set Identifier (SSID) and Media Access Control (MAC) address of a legitimate AP [16]. The MAC is sometimes referred to as Basic Service Set Identifier (BSSID). Another approach is to broadcast the SSIDs of the wireless location a user is connecting to [17], [18], or to broadcast fake AP, which will make it appear as if a user is connected to a Wi-Fi hotspot even if they are not [19]. Attackers then use AP information from a device's preferred network list (PNL) to generate phoney AP tricking devices to connect [20].

For various reasons, including the growing number of users and the sensitivity of the data they hold [6], [21], mobile devices have become the primary target of these assaults. In most business transactions, they have already replaced desktop computers [12]. Since wireless services are available at workplaces and other public places, Internet users prefer wireless connections to get free and convenient Internet service [12], [22]. Furthermore, most users prefer free Wi-Fi over their typical data plans because they do not want to waste their Internet resources [12]. Since users usually connect to any Wi-Fi, their association with illegitimate hotspots poses a high risk. An attacker could intercept data and inject malware into a connected device [22]. These factors increase the chances and risks of Wi-Fi hotspot spoofing [11]. Furthermore, since mobile devices have small computing resources, they do not offer the same level of built-in security mechanisms as desktop computers [12], [23].

Usually, for a mobile device to establish a connection to a wireless AP, it must first perform an active scan which consists of three basic steps, (i) the discovery phase, (ii) authentication, and (iii) association [16]. Initially, a client device sends a request to join the network (probe request), and the AP replies with a probe response. Finally, the client acknowledges and establishes a connection [24]. However, the association process does not distinguish between valid and illegitimate APs in open networks; networks without security protocol running on it. Authentication mechanism used in the association procedure involves exchanging keys with clients associated with them [25]. As a result, clients could connect to any open hotspot due to a desperate need for Internet access. Attackers could take advantage of this flaw by deceiving clients into connecting to their hotspots, and then use that connection to launch more attacks [12], [26].

On the other hand, the network host has a list of devices, and user accounts in authenticated Wi-Fi APs, which are sometimes referred to as closed networks (networks that authenticate clients for Internet access). Users are authenticated based on registered information. Attackers may create a fake hotspot with a fake captive portal to trick network users into achieving authentication procedures. The networks with a captive portal have a web page deployed on the network to authenticate network users at which they are then compelled to submit their login credentials. The login credentials may later be used to launch other attacks in banking or other services involving personal information [12]. Because of the small size of the screen, distinguishing fake captive portals from legitimate ones on mobile phones is much more difficult than on desktop/personal computers (PC) [27]. With a plethora of online tools, such as the Kali Software Engineering Toolkit [28], any web page can be cloned with the same quality as the original with little effort. As a result of this shift, and with a combination of web page cloning and Domain Name System (DNS) poisoning, even desktop users could become victims. Despite efforts to prevent mobile devices from connecting to RAP, spoofing attacks are still possible as a result of mobile users' carelessness with wireless communications [29]. Furthermore, mobile phone users are vulnerable to wireless spoofing attacks due to their fast-on-service behaviour when navigating through services on their phones. This study proposes an Android application prototype that detects hotspot spoofing attacks created by fake wireless hotspots or ETA in wireless networks. The study recognises the importance of preventing spoofing attacks from occurring prior to the mobile device-to-AP association in order to avoid any type of attacks that could be further extended following a successful spoofing attack. The detection could be done by determining the legitimacy of AP using details collected from their broadcasts (probe responses). On the other hand, we adopt the method proposed by [23] to detect fake APs with a captive portal by deceiving the portal with fake login credentials, which minimises the risk of users exposing their true credentials to attackers. We do this by simulating a fake captive portal on Android devices. The captive portal is accessed using Android's WebView package. This work has the following main contributions:

- We propose a prototype for Android devices to detect hotspot spoofing attacks using parameters collected from broadcasting APs in the perimeter.
- We present a solution to detect fake hotspots in networks with the captive portal by deceiving an attacker's captive portal with fake login credentials.
- We implement our prototype through the *FakeAP Detector* on Android-based devices and evaluate its effectiveness based on accuracy and performance.

- We present a testing experiment in three different network structures and compare results with previous studies.

II. RELATED WORKS

This section presents previous works in detecting fake hotspots, upon which the foundation of our work is built—we categorise the review into ETA, fake APs, and mobile-based solutions.

The detection of fake hotspots has been well explored in the existing literature. Ballai [30] presents a system and method for detecting unauthorised APs to wireless networks. The study collects the beacon information from the transmitting AP and determines its validity. The measuring process involves a comparison of the beacon information with a pre-existing database of the communication network. Segura and El-Moussa [31] present methods for authenticating APs as an improvement of [30]. This approach demands that each AP is authenticated first before it is authorised to use the network services. The method has an authentication server, and two identifiers are set on the host network (wired) and wireless device. In addition, an information server is configured with a comparator. If the details of the two matches, then the AP is considered genuine. Bryksa and MacMillan [32] present mechanisms to secure wireless networks using authentication of the wireless client device at the hotspots side. The authentication includes an access controller to establish an encrypted connection between nodes and hotspots. The work by Al-Zubaidie *et al.* [33] presents a lightweight authentication scheme in healthcare applications. Among many other approaches presented, the authors use MAC addresses to verify the legitimacy of authenticating devices. On the other hand, they present an improved approach for server-client authentication to prevent spoofing attacks.

Unfortunately, these approaches rely on the power of the host network. The first work, [30], relies on the database hosted at the network host. The second [31] and third [32] solutions have an authentication server at the host side, which authenticates the client associating with the network. An approach to determine the legitimacy of MAC addresses in [33] relies on the power of the authentication server with a series of operations including, signing the MAC addresses with the PHOTON signatures. This approach demands the authentication server to have a list of legitimate MAC addresses which are signed and the signatures are used during authentication. Since these approaches rely on the power of host network, their implementation in resource-constrained devices like Android devices would significantly affect the performance of the detection ability, which demand high performing algorithms [33].

Matte *et al.* [34] created RAPs to leverage geolocation details available on geotagged services like Facebook and Twitter and simulate other attacks. They also used the BSSID and Received Signal Strength Indicator (RSSI) parameters to predict the location of the AP. BSSID and RSSI make a good combination for identifying spoofed APs. However, if the goal is to identify spoofing attacks on client devices connecting to hotspots, then this strategy is not practical since authentic BSSIDs cannot be identified from end-users, despite the ability to read them. Deshpande and Davenport [35] present a mechanism to detect RAP. It uses messages sent from a user's device. Initially, the user sends the first AP discovery message, including a prestored identifier of previously associated APs. It then sends an additional message, including the identifiers of a non-existent AP. Finally, using a combination of these messages, the device detects the existence of RAPs. However, this approach relies on prestored APs based on previous connections, which might not be helpful when the user is exposed to a new environment. Harmon [36] also presents a system and method to detect illegitimate APs. The approach works on a computer system by (i) detecting when a device attempts to connect to the wireless AP that resembles the legit one, (ii) identifying the location of the computing devices and APs, and (iii) detecting the illegitimate APs by comparing the geographical location of the APs. However, the implementation of the given solution is contingent on the availability of external devices, systems, or network host capacity.

Tchakounté *et al.* [19], who demonstrated how to detect wireless spoofing attacks by evaluating the SSID, BSSID, communication mode, and security protocols, were among the authors influenced our method. Their work assumes that the administrator verifies legitimate APs and that the network administrator knows the entire network and maintains the legitimate, illegitimate, and suspected hosts database. However, Wi-Fi users are typically oblivious of the network's configuration and associated clients. On the other hand, RSSI was not thought to produce accurate detection results in this study. Besides this argument, the RSSI is one of the parameters that attackers cannot duplicate [37], [38] providing a solid foundation for its use in detection algorithms. With the exception of security protocols in closed networks, attackers have complete control over the parameters that govern their functioning. The work by Madani and Vlajic [37], presented the RSSI-based MAC layer spoofing detection using the Deep Learning approach. However, their approach assumes the existence of a large number of scanning nodes which is technically resource-intensive in Android devices since it increases the amount of data collected for comparison. We delved into the idea further, focusing on the client-side, which formed the basis of our work. We begin by looking at Kropeit's work [16], which uses the ''KARMA-Detector'' and the ''Wi-Fi Analyser'' to detect hotspot spoofing on smartphones. The author uses *iw*, and the Wireless Tools software suite, which includes *iwconfig* and *iwlist*, all available in a Linux based environment. The author additionally utilises the *aircrack-ng* suite to crack Wired Equivalent Privacy (WEP) and Wireless Protected Access (WPA), one of the most widely used network auditing software. The detection program sends multiple directed probe requests using randomly generated SSIDs. They did this by using the ''MAC randomiser'' that creates random combinations of SSID and

TABLE 1. Comparison of detection approaches between the proposed system and existing approaches.

Features	[30]	[31]	[32]	[33]	[34]	[35]	[36]	[37]	[38]	[19]	[16]	Proposed solution
Host independence	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✓	✓
Use of Wi-Fi beacons	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
End user focus	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✓	✓
Support for Android device	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Root free	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

BSSID in every probe request. However, the implementation of these methods demands root privilege.

Kim [39] suggests a system and method for detecting RAPs and user devices. The invention uses common beacon characteristics and a database to store beacon parameters. The invention further uses a one-time Uniform Resource Locator (URL) to detect fake hotspots and address resolution protocol. However, with a combination of these parameters, the invention demands a detection server (externally deployed) to incorporate the prescribed elements.

Implementing the majority of approaches [19], [34]–[37] in Android-based devices would increase the communication overhead of the application since limited computing resources already challenge them. On the other hand, the approach in [16] requires a root privilege and has left room for further exploration by capturing and evaluating more packets and getting more information from the packets. For example, AP's security protocol and encryption information were not used. Furthermore, the methods for detecting RAPs and honeypot AP using WEP were left undone. The invention by [39] leaves room for exploration on RAP detection on mobile devices without an external device.

In general, a review of previous research works based on hotspot spoofing detection revealed the following research gaps:

1) The client-side solutions have not explored the identification of fake hotspot spoofing using security information (security protocol and encryption used).
2) Detection approaches in mobile devices are scarce, and those available require root privilege, which is considered dangerous to end-users.

Comparisons between the proposed approach and existing approaches are presented in Table 1. We develop a mobile application prototype (the case of Android) that leverages the advantage of captured hotspot parameters from probe responses to determine their legitimacy. Furthermore, the solution leverages the Wi-Fi beacon frames (SSID, BSSID, RSSI and security information in Android, referred to as capabilities) to detect spoofed hotspots. The probes are captured using Android's built-in functionality, which eliminates the need for external servers and dependence on the capacity of what is known on the host side. We present a portable solution that bridges the gap by eliminating the need for root access while also developing a portable solution for end-users.

III. PROPOSED SYSTEM: FakeAP DETECTOR

We present the *FakeAP Detector* that utilises built-in Android capabilities to detect fake APs on wireless networks based on the mentioned challenges. The solution collects the details of broadcasting APs which are used to determine their legitimacy. On the other hand, the solution challenges the captive portals by submitting randomly generated fake login credentials to the networks that have implemented them. This method also checks if two or more APs are broadcasting with the same SSID and BSSID, which indicates that one or more APs mimics the legitimate AP. These APs are further compared for their similarities in RSSI values and capabilities. RSSI is the intensity of the received signal; the formula to calculate its value is presented in (1). The RSSI is chosen because it is difficult for an attacker to forge [37], [38] and hence increases the chances to identify fake APs. Capabilities store the authentication protocol, encryption algorithm and cipher information of a broadcasting AP.

$$RSSI = TransmitPower + AntennaGain - Pathloss \quad (1)$$

Usually, an attacker's goal in network spoofing is to impersonate the legitimate network to deceive users into connecting to it. Then, they would advance with the Man in the Middle (MiTM) attack. As presented in Fig.1, an attacker may create her network using random SSIDs that reflect the names of the places in the perimeter. For instance, if an attack is simulated at university premises, an attacker could name her APs after the class names, hostel buildings, laboratories etc. But, on the other hand, she could create APs with convincing SSID such as Free Wi-Fi, Student-Wi-Fi, Student 5G Access and the like. Alternatively, an attacker could simulate an ETA based on the details of a legitimate network or de-authenticate users from the network they are currently connected to using

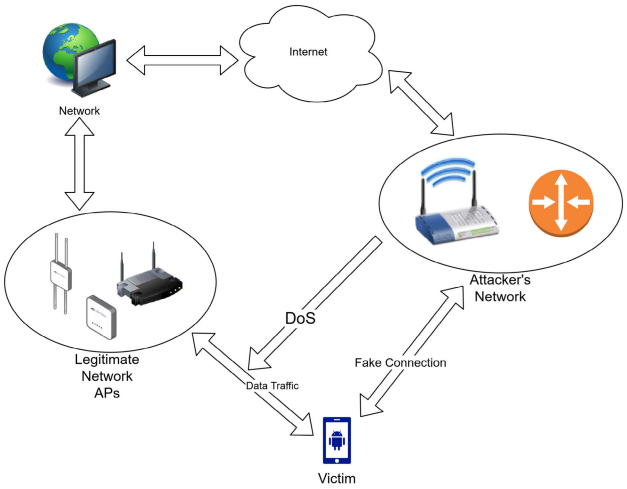


FIGURE 1. Attack threat model.

simple tools available in Kali Linux Operating System (OS) like *aircrack-ng*. These networks would not have any noticeable differences to end-users, so they would connect to them if they have internet access.

A. SCOPE AND ASSUMPTIONS

We present our solution focusing on Android-based devices. However, it might also be feasible for other mobile devices as the challenges are similar, and users of most mobile device OSs in the market have demonstrated similar behaviours [40]. Furthermore, since our solution aims to use the built-in Android phone features and development tools, few features were extracted from broadcasting APs in the perimeter due to the Android OS limitations. More features could be extracted with the help of an external device or by rooting the device. However, the few features we collected were sufficient to develop the *FakeAP Detector*. The presented prototype detects the presence of fake APs which broadcasts during the scanning process period.

On the other hand, in the networks with captive portals, we have assumed that a captive portal pops up automatically after a user is connected to a network. The designed captive portal is meant to read usernames and passwords from users. The Hypertext Transfer Protocol (HTTP) responses could be used to determine the legitimacy of broadcasting AP and its captive portal. Throughout the exercise of scanning for broadcasting APs and determining their legitimacy, the devices involved in the process are assumed to be in a static position.

As we have discussed in Section I, network spoofing attacks exist in various forms. Some behave differently based on tools used to simulate attacks. This study did not cover spoofing attacks created by deceiving SSIDs that do not imitate the features of the legitimate AP. The study assumes further that, an attacker creates fake APs mimicking AP details from the legitimate network since it is unlikely for an attacker to simulate ETA before the legitimate APs are broadcasted.

B. SYSTEM MODEL

Wireless networks do not encrypt its transmissions during communication, so wireless cards could easily capture details of APs in the perimeter. Attackers utilise this benefit to mimic legitimate networks and harvest personal information from users connecting to their networks. The detection measures are of no difference. They utilise the ability to capture wireless frames from APs and determine their legitimacy. We first discuss methods for detecting network spoofing attacks based on Evil Twin Attack (ETA), and second, we discuss methods for detecting fake captive portals in a wireless network.

C. DETECTION OF SPOOFING ATTACKS

With the built-in Android Wi-Fi card we were able to capture details of the broadcasting APs. Some that we were able to capture and use for our detection prototype are SSID, BSSID, capabilities and RSSI. The capabilities include encryption referred to as ENC, CIPHER, and authentication type referred to as AUTH. These details help us detect attacks in which an attacker mimics features of legitimate APs. For example, most attacks targeting wireless networks using *aircrack-ng* suite, mimic SSID, MAC, Channel, and others but cannot mimic RSSI since they are generated based on the power of AP. So, they cannot be manipulated by adversaries in wireless networks [37], [38], [41]. Additionally, despite the possibility of creating an ETA with WEP, WPA or WPA2 in authenticated APs, most attackers ignore capabilities information because they want to leave their network open for clients to associate.

The *FakeAP Detector* starts by scanning for available APs in the perimeter. The details are then stored in the database as they are captured. When the scan is complete, the AP details are retrieved from the database and compared for similarities. If two or more APs broadcast with the same SSID and BSSID, then one or more among them is questionable. To further be sure, the remaining two features, RSSI and capabilities of the twin-APs, are compared. If there are differences in RSSI or capabilities, it is confirmed that fake AP(s) exist in the perimeter. Unfortunately, the RSSI value is not constant and is affected by several environmental factors such as obstacles [37]. Furthermore, capabilities information may not return the desired result since both legitimate and illegitimate APs would broadcast with the same capabilities information. To address these challenges, we use the difference in the means of RSSI values to detect fake AP. The suggested solutions are classified into two categories based on network characteristics: detection in open networks and detection in closed networks.

According to data captured from an Android phone during our experiments in an open network, we observed that these networks do not employ any security mechanisms. As a result, both legitimate and bogus networks may have similar capabilities because an attacker typically creates an open network to which anyone can connect. In this case, the solution is to rely on the RSSI value, which is not static. The problem could be solved by scanning for broadcasting APs in multiple rounds, resulting in a cluster of RSSI values for comparison. The *FakeAP Detector* scans in ten rounds (the rounds could be adjusted to any number depending on desired speed and accuracy). The results were stored in the database after each round. The solution works by comparing the average of RSSI values of each duplicate broadcasting APs to the average of RSSI value of the first broadcasting AP among the duplicates. Because an ETA imitates the original networks, the second broadcasting AP among the twin-APs with a different average RSSI value could be a fake.

To work on this, we benchmark the RSSI value of the first AP to broadcast among the duplicate APs. The benchmarked value will then be used to retrieve the RSSI values from the same SSID, which fall in the range of $(+5$ and $-10)$ dBm. The difference of the means for the collected RSSI values from duplicate APs will be calculated. If the difference exceeds three units of dBm, then one of the duplicate APs is fake. This range is defined based on RSSI fluctuations

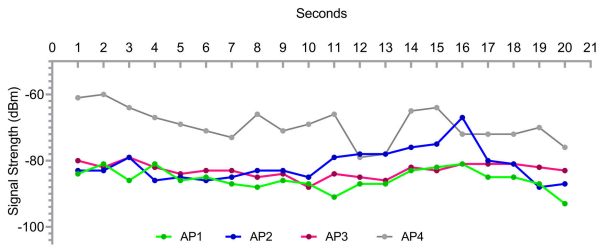


FIGURE 2. Signal strength fluctuations along with time in seconds.

captured in the experimental observation, as presented in Fig. 2. The scanning was simulated in twenty rounds, each round capturing broadcasts in one second.

For instance, with AP1, the benchmarked value is −84 dBm, at which the strongest signal was −81 dBm, and the weakest was −93 dBm making a difference of (+3 and −9) dBm in the high and low signal fluctuations, respectively. The AP2 had a difference of (+16 and −4) dBm from its benchmarked value of −83. The AP3 had a difference of (+1 and −8) dBm. Lastly, the AP4 had a difference of (+1 and −19) dBm, as presented in Table 2. The highest and lowest signal strength range was calculated to be 12, 20, 9 and 19 for the AP1, AP2, AP3 and AP4, respectively. These make an average range of 15 units of dBm to which signal strength could fluctuate. Our study used the +5 and −10 difference to select RSSI values to calculate the mean for a specific AP. This threshold falls into 15 dB units of the range determined in our experimental study.

TABLE 2. The presentation of benchmark RSSI value, highest signal, lowest signal and the range between highest and lowest signal.

Values	AP1	AP2	AP3	AP4
Benchmarked value	-84	-83	-80	-61
Highest signal	-81	-67	-79	-60
Lowest signal	-93	-87	-88	-79
Range (High - Low)	12	20	9	19

In closed networks, the detection is straightforward under the assumption that attackers create open fake APs for everyone to connect. When this is the case, all closed networks broadcast with security information indicating the protocol and encryption used. Contrary, an open network would broadcast without indicating security protocols and encryption used. Hence, to detect fake APs, we first check for twin-AP, if they exist, then capabilities information is compared. When the difference is noticed, the AP without security protocol and encryption information and the one which came after the other had started broadcasting is marked as fake. In cases where an attacker creates an ETA with security information similar to legitimate AP, the comparison of RSSI values is done on top of capabilities (security) information to increase efficiency in detecting fake APs. The detection flow is presented in Fig. 3.

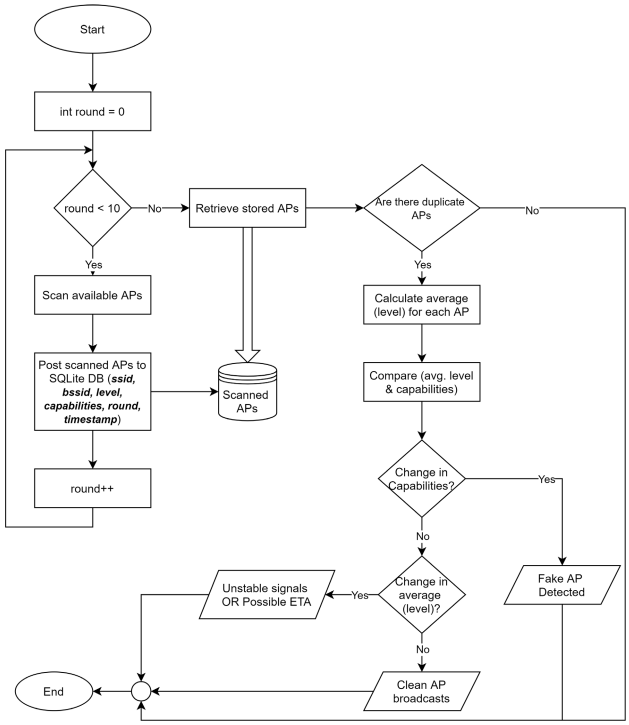


FIGURE 3. Fake AP detection flowchart.

D. NETWORKS WITH CAPTIVE PORTAL

In this context, the solution is built under the foundation of web protocols and Android's *WebView* class. In other words, the detection relies on response messages from Hypertext Transfer Protocol (HTTP). Two HTTP responses are considered: the login attempt failure message and the success message. During authentication on the web, error message 401 is given for unauthorised users and 403 for forbidden requests [23]. Then, response code 200 is given when the submitted request has succeeded. Therefore, we focus on error response 401 captured by the *WebResourceResponse* class of the *WebView* package.

To determine the legitimacy of the network, the *FakeAP Detector* generates random login credentials using the JavaScript code which follows the pseudo code presented in Algorithm 1. The generated credentials are then submitted

Algorithm 1 Login Automation on the Fake Captive Portal

Input: *username* ← *randomString*(10)
Input: *password* ← *randomString*(10)
1: Load captive portal URL
2: **if** *pageLoaded* **then**
3: Input username
4: Input password
5: Submit credentials
6: **else**
7: **return** *Error*
8: **end if**

into a captive portal. Next, the response codes of the portal are determined. The input lines of the algorithm generates random username and password. Then, the first and second lines load the captive portal and determine its status using web response codes. The third and fourth lines auto-fill the form inputs by identifying form input names and submitting values generated in the input lines. Lastly, the values are automatically submitted to the captive portal web server in the fifth line. After submission, when the success code is returned, or no error messages are received in the process, then the captive portal originates from a fake network (See Fig. 4).

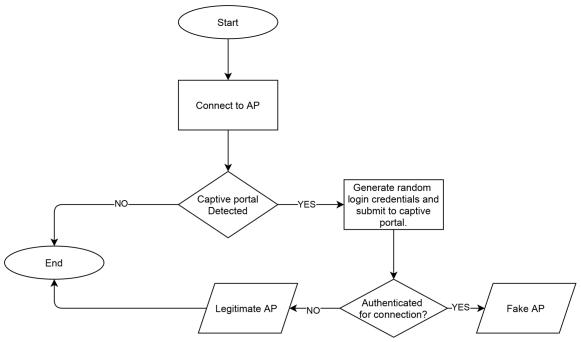


FIGURE 4. Fake captive portal detection flowchart.

IV. MATERIALS AND METHODS

In this section, we present materials and methods used in developing the *FakeAP Detector*. In addition, methods and tools used in the two experiments are also presented in two different network structures with consideration of attacks in open and closed networks.

A. EXPERIMENTAL DESIGN

We created a network of fake APs with three considerations. First, we considered a set of APs that mimics the characteristics of an open legitimate AP. Second, a set of APs that mimics the details of a closed AP and third, a set of APs that deceive clients with fake AP and a fake captive portal (captive portals presented in Fig. 5). The first and the second broadcasts were captured on Android devices and a personal computer (PC). The packets in a PC were captured using Wireshark, and two approaches were used in Android devices. First, we used the Android Pcap library, which generate pcap files that can be analysed using network analysis tools. Second, we used WifiManager Java classes to capture AP details. The captures in both devices were observed for noticeable differences between legitimate AP and fake APs. Fig. 6 shows sample results of captures in the *FakeAP Detector*. The details in Fig. 6 (a) presents the APs scan results in a series of rounds.

On the other hand, fake APs with fake captive portals were created to mimic the captive portal of the legitimate network. The fake captive portal was created in basic HyperText Markup Language (HTML), Cascading Style Sheet (CSS) and JavaScript (JS) with PHP running in the backend and hosted at our own domain available at *http://fakeap.lunodzo.com/*. The server was running under the *phpMyAdmin* tool, which supports PHP and MySQL. Using the Android's *WebView* packages, we could view the fake captive portal in Android as presented in Fig. 7. Mzumbe University was chosen for this exercise, as they have implemented a captive portal on their network.

A NETGEAR Nighthawk®X6 AC4000 Tri-Band Wi-Fi router was used for experiments. It is a tri-band Wi-Fi providing three (3) dedicated bands that are optimised for speed. These features allowed us to broadcast up to six (6) APs simultaneously with distinct SSIDs and configure them to be open or closed. The router was useful in all experiments since we used it to create a network of APs marked as legitimate in our experiments.

We simulated an attack in our lab environment to study visible patterns used in developing the *FakeAP Detector*. The experimental setup involved two PCs, one installed with the Kali Linux 2021.2 and another with Ubuntu 21.04. The study used the Kali Linux 2021.2 to simulate the attacks with the help of the Network Interface Card (NIC) that supports monitor mode. The Alfa One AWUS036H 1000mW Chipset RTL8187L was used. The PC running Ubuntu 21.04 OS had a built-in wireless chipset running in monitor mode as well. A list of commands used to simulate an attack is presented in Table 3. Initially, the Wi-Fi chipset was set into monitor mode using the *airmon-ng* command. The second step was to check the broadcasting APs in the perimeter using *airodump-ng*. The command enables the collection of AP details that could be used to simulate an attack. Finally, the *airbase-ng* command created an evil twin AP based on details collected in the second step.

TABLE 3. List of commands for wireless ETA.

Command	Function
airmon-ng	Set Wi-Fi adapter into monitor mode.
airodump-ng	Scan broadcasting APs
airbase-ng	Attacking clients as opposed to the AP (ETA)
aireplay-ng	Used to inject packets, de-authentication attack

The second PC was used to capture packets in pcap format and then analyse them using the Wireshark.[1] The tool helps network administrators to examine the live network data or saved pcap files for troubleshooting network traffic or detecting potential malicious activities.

B. APPLICATION DEVELOPMENT

We employed the Extreme Programming (XP) development methodology to accommodate the constantly changing application requirements. The implementation of the *FakeAP Detector* is based on Android-based development tools. We used the Android Studio as the Integrated Development Environment (IDE). The backend implementation was based

[1] Wireshark

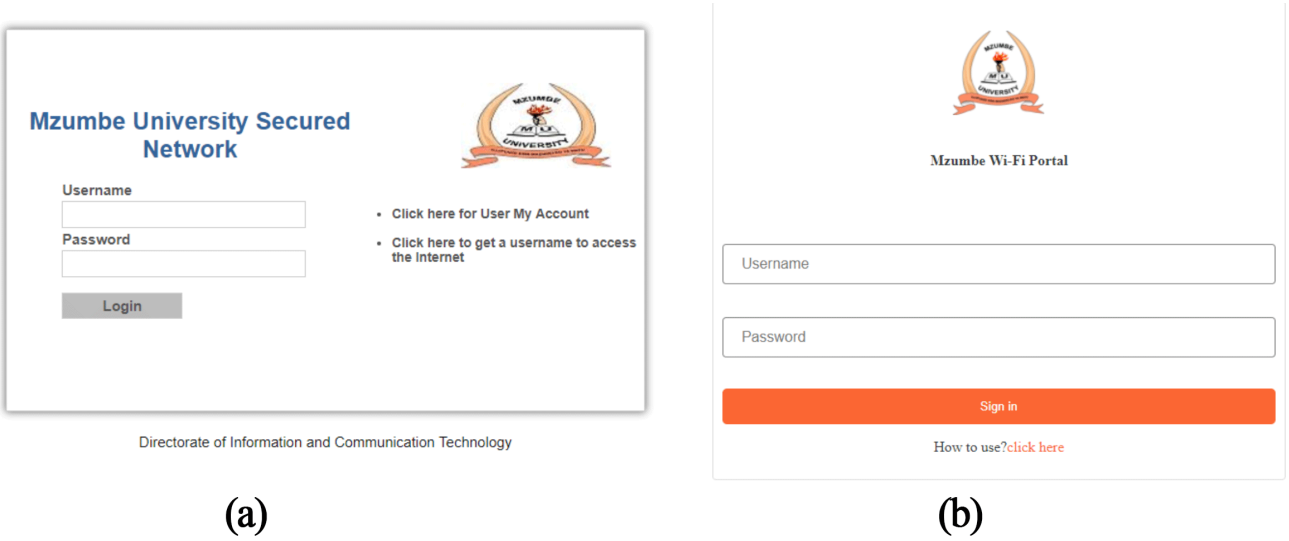


FIGURE 5. The captive portals screenshot. (a) Legitimate captive portal and (b) Fake captive portal.

FIGURE 6. (a) AP scan results and (b) Logged pcaps.

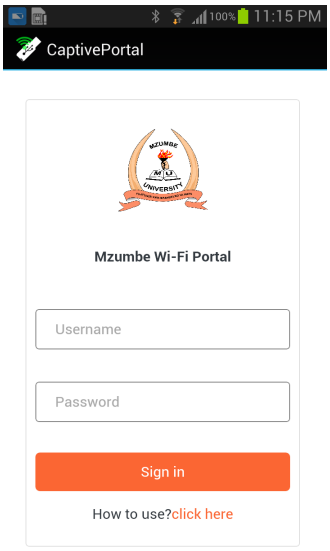


FIGURE 7. Fake Captive portal view in the *FakeAP Detector*.

on the Java programming language, and the data were stored in the Android's SQLite database. The application's interfaces were developed using the eXtensible Markup Language (XML), making the application light and fast since XML is a lightweight language and would not make our layout heavy.

Furthermore, the Android PCAP library[2] from Kismet was used to manage the packet capture within an application with the help of the Alfa One wireless card. The Android PCAP implements the Linux kernel RTL8187 driver using the Android USB host API; hence it doesn't require root privileges. Furthermore, the Alfa One 802.11b/g wireless card implements the RTL8187 drivers, making it suitable for our experiments.

The SAMSUNG I9300 Galaxy SIII was used to deploy and test an application. It supports up to Android 4.4, KitKat GT-I9301I Neo only. In addition, the Galaxy SIII works perfectly with the Android PCAP library and supports USB host operation, allowing us to use an external NIC to capture raw packets.

The Alfa One NIC was plugged into the Android device with the help of an On-The-Go (OTG) cable to capture raw pcap files. The captures were meant to be parsed in our detection prototype to determine their legitimacy. Figure 6 (b)

[2]Android Pcap library

shows a screenshot of scan results of pcap files. The *io.pkts*[3] was used to manipulate the captured packets. It is a package capable of reading and writing from the pcap files. Figure 8 presents the flow of packet captures. Initially, packets from the router are captured in our detection application with the help of the Alfa One NIC. Then, the packets are handled by the Android PCAP library.

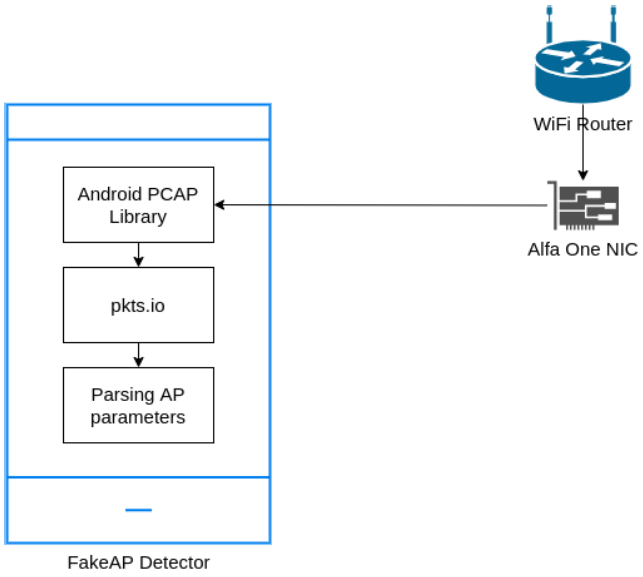


FIGURE 8. The flow of pcap files in *FakeAP Detector*.

V. EXPERIMENTAL SETUP AND RESULTS

This section presents the results obtained during the testing and deployment of the *FakeAP Detector*.

A. AP DETAILS CAPTURE

The initial step towards the detection of fake APs includes scanning active APs broadcasting in the perimeter. We did this in two ways, first by using an external NIC and second by using the Android's built-in NIC. The scan results with an external NIC were presented in Fig. 6 (b). Each session of scan stores the packets in a different file. This process was adapted from the Android PCAP Library available in the Google Play store. Figure 6 (a) shows the scan results using the built-in wireless card to which SSID, BSSID, RSSI, and capabilities were captured and stored in the SQLite database.

We could not work further with the raw pcap files due to the limitation of the *io.pkts*. After the capture, we were supposed to filter packets relevant for our detection application, i.e., the probe response from APs. Unfortunately, the *io.pkts* did not support parsing of the 802.11 protocol as of August 2021.

B. DATA STORAGE

The AP information captures were stored in a single table of the SQLite database. The designed table covered basic information from the AP and added a few, which helps identify scans uniquely. Sample scan results are shown in Fig. 9 as stored in the database. The probes are scanned in ten rounds (the number of rounds could be adjusted to any) to capture different RSSI values. Each round is then posted into a database table. When the scan rounds are completed, the detection process starts. The detection method retrieves data from the database and uses the details to compare each AP's legitimacy based on the algorithm presented in Fig. 3. To avoid overloading an application with useless data, the database is wiped during the start of every scanning process.

id	ssid	bssid	level	capabilities	round	comment	time
F...	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	kisarouter	78:54:2e:92:2f:ce	-59	[WPA2-PSK-CCMP][WPS][ESS]	1	0	22:40:27.088
2	NMAIST-CDAC 17	14:02:ec:4c:2c:90	-62	[WEP][ESS]	1	0	22:40:27.119
3	VLIR_L CLASES	dc:9f:db:62:a5:a0	-73	[ESS]	1	0	22:40:27.123
4	kisarouter	78:54:2e:92:2f:ce	-62	[WPA2-PSK-CCMP][WPS][ESS]	2	0	22:40:28.753
5	NMAIST-CDAC 17	14:02:ec:4c:2c:90	-64	[WEP][ESS]	2	0	22:40:28.780
6	VLIR_L CLASES	dc:9f:db:62:a5:a0	-73	[ESS]	2	0	22:40:28.785
7	kisarouter	78:54:2e:92:2f:ce	-63	[WPA2-PSK-CCMP][WPS][ESS]	3	0	22:40:30.404
8	NMAIST-CDAC 17	14:02:ec:4c:2c:90	-62	[WEP][ESS]	3	0	22:40:30.422
9	VLIR_L CLASES	dc:9f:db:62:a5:a0	-72	[ESS]	3	0	22:40:30.427

FIGURE 9. Sample scan results (three rounds) as it can be seen in the SQLite database.

C. FAKE AP DETECTION

Since the application collects information of the broadcasting APs at a certain period and then stores them into a database, the detection prototype fetches the scanned details from the database. First, we use Structured Query Language (SQL) statements to determine duplicate APs in each round based on SSID and BSSID information. These statements are presented in Listings 1, 2 and 3. The duplicate APs are then compared with their details to determine their legitimacy based on the flowchart shown in Fig. 3. This process was categorised into open networks and closed networks. In closed networks, usually, the legitimate network has WEP, WPA or WPA2 enabled. The first line into the detection approach starts with comparing the capabilities information. The SQL statement to fetch duplicate APs with different capabilities is presented in Listing 1.

```
SELECT ssid, bssid FROM accesspoints
GROUP BY ssid, bssid HAVING
min(capabilities) <> max(capabilities)
```

LISTING 1. The SQL statement that returns duplicate APs with different capabilities, abovecaptionskip=0pt, belowcaptionskip=0.

An attacker usually has most of the details similar to legitimate AP in open networks, including the capabilities. In this case, we focus on RSSI. We created a temporary table (view) in the database where AP details are stored. The view stored all duplicate APs whose capabilities, SSID and BSSID, were the same. The implementation is presented in Listing 2.

From the list of duplicate APs, we compare their RSSI values determining the legitimacy of each AP. Listing 3

[3]io.pkts package

```
CREATE TEMP view duplicateCapabilities AS SELECT
    ssid, bssid, capabilities, level, time FROM
    accesspoints a1 WHERE EXISTS (SELECT 1 FROM
    accesspoints a2 WHERE a1.ssid = a2.ssid AND a1
    .bssid = a2.bssid AND a1.capabilities = a2.
    capabilities AND capabilities = ESS) EXCEPT
    SELECT ssid, bssid, capabilities, level, time
    FROM accesspoints a1 WHERE EXISTS(SELECT 1
    FROM accesspoints a2 WHERE a1.ssid = a2.ssid
    AND a1.bssid = a2.bssid AND a1.capabilities !=
     a2.capabilities)
```

LISTING 2. The SQL statement that creates a view that stores duplicate open APs.

```
SELECT * FROM duplicateCapabilities dp1 WHERE
    EXISTS (SELECT 1 FROM duplicateCapabilities
    dp2
WHERE dp1.ssid = dp2.ssid AND dp1.bssid = dp2.
    bssid AND dp1.capabilities = dp2.capabilities
    AND
average_level NOT BETWEEN (storedFirstSignal-10)
    AND  (storedFirstSignal+5))
```

LISTING 3. The SQL statement that returns results of APs with average levels not falling in the defined range.

presents the SQL statement, which determines the difference in average RSSI values based on the benchmarked RSSI value. The duplicate APs with average RSSI values not falling in a defined range are marked as fake.

D. TESTING

We have designed three experiments following the designed attack detection methods developed. The first two setups involve the testing approaches for the fake APs based on open and closed network structures. The third setup was for the network that has implemented a captive portal to authenticate clients. As presented in Fig. 10, these experiments used a wireless router, an Android device, and a PC. The attacking device had a NIC supporting monitor mode giving it the ability to read parameters of APs in the perimeter and simulate an ETA. The NETGEAR router could simultaneously broadcast up to six (6) APs with different configurations. The tests were simulated assuming a constant position of an adversary (attacking device), legitimate APs and the detecting device.

In the first experiment, we broadcast APs (legitimate and illegitimate) in a closed network. Thus, the illegitimate network of APs was mimicking the legitimate network. Similarly, we broadcast an open network with its details imitating the legitimate network in the second experiment. To obtain reliable results, we did a hundred test experiments, and in each, we calculated accuracy and detection speed. Finally, the average of each performance indicator was obtained.

In the third experiment, we simulated a network with the captive portal. The fake captive portal was then tested with a fake credential in the *FakeAP Detector*. Finally, the HTTP response was captured to determine the legitimacy of the captive portal.

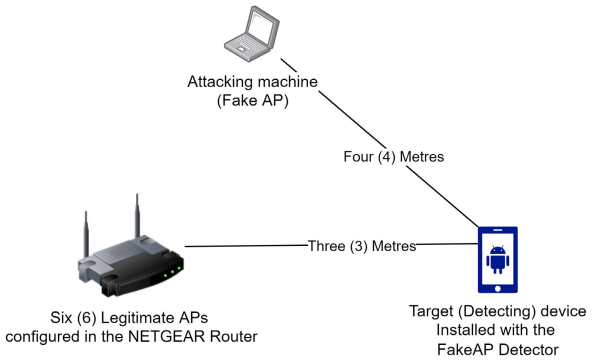


FIGURE 10. Experimental setup topology.

On the other hand, we looked at how the number of rounds could affect the performance of the detection prototype. We tested the prototype by collecting AP details in two (2), ten (10), and twenty (20) and fifty (50) rounds. The lower number of rounds would reduce the time taken for scanning and detection (better speed), however, it would affect the detection accuracy. Contrary, scanning in many rounds would negatively impact the detection time and speed but increase the detection accuracy. For instance, scanning in two rounds would take 1.5 seconds in scanning and storage. On the other hand, scanning in ten rounds would take 11 seconds, while scanning in twenty and fifty rounds would take 17 and 48 seconds. Similarly, the detection in two rounds scan results would spend an average of 2 milliseconds with an accuracy of 40%. Ten rounds would spend an average of 24.98 milliseconds, while twenty and fifty rounds would spend an average of 97 milliseconds and 2 seconds, respectively, with an accuracy of 99.6% for twenty rounds 99.8% for fifty rounds. Ten rounds were chosen considering the trade-off between the two metrics, as detection time is important in Intrusion Detection Systems (IDS). Our experiments observed that adding the number of rounds significantly affects detection time with little improvement in detection accuracy.

E. EXPERIMENTAL EVALUATION

To obtain reliable results and build evident conclusions from the detection prototype, we run the scans and detection in a clean and attacked environments to see the test scores. The experiment shows 2% and 1% false positives in open and closed networks respectively and 98% and 99% true negatives in open and closed networks respectively, as presented in Table 4 and 5. During the tests in clean environment, the average detection speed of the prototype was calculated to be 24.98 milliseconds when AP scan results were already stored in the database. We noticed further that; time spent in detection is affected by the number of broadcasting APs. The presented results are based on a network with a total of seven broadcasting APs in the perimeter.

The same experiment was conducted in a network with an attack. In this case, we considered two different attacks: an attack targeting open APs and a second attack targeting

TABLE 4. Detection test results in open network.

Condition	Accuracy in %
True positives (Attack present and detected)	99%
False positives (Attack not present and detected)	2%
True negative (Attack not present and not detected)	98%
False negative (Attack present and not detected)	1%

TABLE 5. Detection test results in closed network.

Condition	Accuracy in %
True positives (Attack present and detected)	99.7%
False positives (Attack not present and detected)	1%
True negative (Attack not present and not detected)	99%
False negative (Attack present and not detected)	0.3%

closed APs. In the open APs, the results show that the *FakeAP Detector* has achieved 99% true positives and had 1% false negatives, as presented in Table 4 with an average of 24.64 milliseconds of detection time. This experiment had seven broadcasting APs with one being fake.

In a closed network, the results show that the *FakeAP Detector* has achieved 99.7% of true positives and had 0.3% of false negatives with an average of 5.78 milliseconds of running time. The detection time in this attack was significantly low since the algorithm first checks for differences in security information. If the difference is noticed, then the process ends (refer to Fig. 3). The detection accuracy for this experiment is shown in Table 5.

In the captive portal, our detection accuracy was 88% in one hundred tests done. The captive portal reads and returns a success message immediately after the credential is successfully posted into a database without verifying the details. Here, the performance was highly affected by the availability of the Internet and the server where the fake captive page was hosted.

These results were also calculated to obtain precision, recall and F1-Score metrics scores. The *FakeAP Detector* achieved a 98% precision, 99% recall and 98.4% F1-score in open networks. On the other hand, it has achieved 99% precision, 99.7% recall and 99.3% F1-score in a closed network. These performances are competitive compared to similar recent studies, including the work by Madani and Vlajic [37] as presented in Table 6.

TABLE 6. Performance comparison between the FakeAP Detector and Deep Learning Detection approach.

	FakeAP Detector		Deep Learning	
	Open network	Closed network	Day Classifier	Night Classifier
Precision	0.980	0.99	0.97	0.99
Recall	0.990	0.997	1.0	1.0
F1-Score	0.984	0.993	0.98	0.99

VI. LIMITATIONS AND FUTURE VENTURES

Android OS is a circular improving OS. Various improvements are made in each of their releases. This makes it a challenge for researchers to develop sustainable solutions focusing on Android-based devices. Nevertheless, the studies have the opportunity to contribute to the body of knowledge. This study was conducted in the spirit of contributing to the body of knowledge. So far, we have covered the detection of fake APs that mimic legitimate networks. However, fake APs created with random APs or those that do not mimic the structure of legit networks were not covered in this study, and they would demand a different approach to be detected. This work did not manipulate details captured in pcap files; this would result in more information relevant for detecting fake APs by comparing legitimate APs from the fake APs packets. As a remedy, future works may need to redevelop their classes following the guide provided *io.pkts* documentation, which could result in rich features, including vendor-specific information and round-trip time, which could strengthen the detection effectiveness.

Our detection approach using RSSI values may sometimes result in false positives if legitimate and fake APs broadcast with similar RSSI values, especially in open networks. Therefore, this approach has to be used in combination with other parameters collected from the broadcasting APs. On the other hand, an attacker could create an ETA with features similar to legitimate AP, including security information. In this case, the prototype relies on RSSI value only.

Determining the estimated location of legitimate APs using the RSSI value also leaves a promising research gap. Since RSSI is one of the features that an attacker cannot mimic, a strong fake AP detection solution could be developed based on an AP's location. The detection of fake AP that de-authenticates clients and lets them connect to their network was not done due to limited features collected from the broadcasting APs. A pcap file could easily help since the packets are classified based on their types, including the de-authentication packets. Furthermore, features currently available in 802.11ax, such as Basic Service Set (BSS) colouring, also leave a promising possibility for detecting fake APs. The study invites research work to build a prevention system for network spoofing attacks.

The presented solution is subjected to possibilities of false fake AP detection results, this happens when the detection measures rely on the difference in average of RSSI values of duplicate APs. There are trade offs in balancing the range to be used to determine legitimacy of APs. The smaller range puts the prototype in a good detection accuracy but with high possibility of false positives. On the other hand, large range may solve the problem of false positives but increases the chances of missing out the existence of fake APs.

VII. CONCLUSION AND DISCUSSIONS

This paper proposed a prototype of an Android application to detect hotspot spoofing attacks in wireless network settings using features collected from broadcasting APs. The proposed prototype collects SSID, BSSID, RSSI and capabilities information which are being compared. The comparison decisions are made depending on the nature of the network.

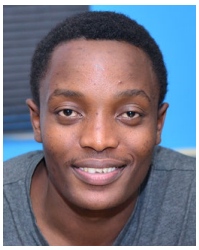
Initially, AP is compared for the differences in capabilities in closed networks, and later RSSI value is used when capabilities information appears to be the same. In contrast, a combination of capabilities and RSSI values is used in open networks since both legitimate and illegitimate APs broadcast similar capability information.

On the other hand, our method challenges the networks with a fake captive portal by submitting fake login credentials to test their legitimacy. We analysed the performance of the *FakeAP Detector*, which had an accuracy detection of 98% and 99% in open and closed networks, respectively. The prototype had shown the best detection time when detecting in a closed network where an average of 5.78 milliseconds was spent. In the open network, an application had spent an average of 24.64 milliseconds in detection. The development of the prototype focused on a lightweight solution on Android-based devices; hence using the built-in Android resources yields better results and performance without rooting the device. We had used the *WifiManager*, *WebView* and *SQLite* packages to make this possible.

REFERENCES

[1] J. Johnson. (2021). *Global Digital Population as of January 2021*. Accessed: May 25, 2021. [Online]. Available: https://www.statista.com/statistics/617136/digital-population-worldwide/
[2] S. O'Dea. (2021). *Number of Smartphone Users From 2016 to 2021*. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[3] A. Joyce-Gibbons, D. Galloway, A. Mollel, S. Mgoma, M. Pima, and E. Deogratias, "Mobile phone use in two secondary schools in Tanzania," *Educ. Inf. Technol.*, vol. 23, no. 1, pp. 73–92, Jan. 2018.
[4] W. Suryasa, J. R. Z. Mendoza, T. M. Mera, M. E. M. Martinez, and M. R. Gamez, "Mobile devices on teaching-learning process for high school level," *Int. J. Psychosoc. Rehabil.*, vol. 24, no. 4, pp. 330–340, Feb. 2020.
[5] L. Einav, J. Levin, I. Popov, and N. Sundaresan, "Growth, adoption, and use of mobile E-commerce," *Amer. Econ. Rev.*, vol. 104, no. 5, pp. 489–494, 2014.
[6] R. Bitton, A. Finkelshtein, L. Sidi, R. Puzis, L. Rokach, and A. Shabtai, "Taxonomy of mobile users' security awareness," *Comput. Secur.*, vol. 73, pp. 266–293, Mar. 2018.
[7] Cisco. (Mar. 2020). *Cisco Annual Internet Report (2018–2023) Report*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
[8] L. Lazos and M. Krunz, "Selective jamming/dropping insider attacks in wireless mesh networks," *IEEE Netw.*, vol. 25, no. 1, pp. 30–34, Jan. 2011.
[9] D. Kidston and L. Li, "Management through cross-layer design in mobile tactical networks," in *Proc. IEEE Netw. Oper. Manage. Symp.*, Osaka, Japan, Apr. 2010, pp. 890–893.
[10] C. Wolfe, S. Graham, R. Mills, S. Nykl, and P. Simon, "Securing data in power-limited sensor networks using two-channel communications," in *Proc. Int. Conf. Crit. Infrastruct. Protection*. Arlington, VA, USA: Springer, 2018, pp. 81–90.
[11] K. Jindal, S. Dalal, and K. K. Sharma, "Analyzing spoofing attacks in wireless networks," in *Proc. 4th Int. Conf. Adv. Comput. Commun. Technol.*, Washington, DC, USA, Feb. 2014, pp. 398–402.
[12] Kaspersky. *Top 7 Mobile Security Threats in 2020*. Accessed: May 30, 2021. [Online]. Available: https://www.kaspersky.com/resource-center/threats/top-seven-mobile-security-threats-smart-phones-tablets-and-mobile-internet-devices-what-the-future-has-in-store
[13] J.-M. Seigneur, "Wi-trust: Computational trust and reputation management for stronger hotspot 2.0 security," *J. ICT Standardization*, vol. 4, no. 3, pp. 213–236, 2017.
[14] P. Shrivastava, M. S. Jamal, and K. Kataoka, "EvilScout: Detection and mitigation of evil twin attack in SDN enabled WiFi," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 1, pp. 89–102, Mar. 2020.
[15] V. B. Srinivas and S. Umar, "Spoofing attacks in wireless sensor networks," *Int. J. Sci., Eng. Technol., Comput.*, vol. 3, no. 6, p. 201, 2013.
[16] T. Kropeit, "Don't trust open hotspots: Wi-Fi hacker detection and privacy protection via smartphone," M.S. thesis, Dept. Embedded Secur., Ruhr-Universität Bochum, Bochum, Germany, 2015.
[17] M.-W. Park, Y.-H. Choi, J.-H. Eom, and T.-M. Chung, "Dangerous Wi-Fi access point: Attacks to benign smartphone applications," *Pers. Ubiquitous Comput.*, vol. 18, no. 6, pp. 1373–1386, Aug. 2014.
[18] K. Wang, S. Chen, and A. Pan, "Time and position spoofing with open source projects," *Black Hat Eur.*, vol. 148, pp. 1–8, Nov. 2015.
[19] F. Tchakounte, M. Nakoe, B. O. Yenke, and K. P. Udagepola, "Recognizing illegitimate access points based on static features: A case study in a campus WiFi network," *Int. J. Cyber-Secur. Digit. Forensics*, vol. 8, no. 4, pp. 279–291, 2019.
[20] G. Chatzisofroniou. *The Known Beacons Attack (34th Chaos Communication Congress)*. Accessed: May 27, 2021. [Online]. Available: https://census-labs.com/news/2018/02/01/known-beacons-attack-34c3/
[21] J.-S. Oh, M.-W. Park, and T.-M. Chung, "The multi-level security for the Android OS," in *Proc. Int. Conf. Comput. Sci. Appl.* Cham, Switzerland: Springer, 2014, pp. 743–754.
[22] NortonLifeLock. (2019). *Why Hackers Love Public Wi-Fi*. Accessed: Sep. 30, 2010. [Online]. Available: https://us.norton.com/internetsecurity-wifi-why-hackers-love-public-wifi.html
[23] J. D. Ndibwile, Y. Kadobayashi, and D. Fall, "UnPhishMe: Phishing attack detection by deceptive login simulation through an Android mobile app," in *Proc. 12th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2017, pp. 38–47.
[24] D. Jaisinghani, G. Singh, H. Fulara, M. Maity, and V. Naik, "Elixir: Efficient data transfer in WiFi-based IoT nodes," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2018, pp. 823–825.
[25] M. A. E. I. Mohammad, "Wireless LAN security (IEEE 802.11b)," M.S. thesis, Dept. Comput. Sci. Eng., BRAC Univ., Dhaka, Bangladesh, 2008.
[26] M. Bernaschi, F. Ferreri, and L. Valcamonici, "Access points vulnerabilities to DoS attacks in 802.11 networks," *Wireless Netw.*, vol. 14, no. 2, pp. 159–169, Apr. 2008.
[27] J. D. Ndibwile, E. T. Luhanga, D. Fall, D. Miyamoto, G. Blanc, and Y. Kadobayashi, "An empirical approach to phishing countermeasures through smart glasses and validation agents," *IEEE Access*, vol. 7, pp. 130758–130771, 2019.
[28] N. Pavković and L. Perkov, "Social engineering toolkit—A systematic approach to social engineering," in *Proc. 34th Int. Conv. MIPRO*, 2011, pp. 1485–1489.
[29] R. Prasad and V. Rohokale, "Mobile device cyber security," in *Cyber Security: The Lifeline of Information and Communication Technology*. Cham, Switzerland: Springer, 2020, pp. 217–229.
[30] P. N. Ballai, "System and method for detection of a rogue wireless access point in a wireless communication network," Jun. 27, 2006, U.S. Patent 7 068 999.
[31] X. J. Segura and F. El-Moussa, "Method and system for authenticating a point of access," Nov. 18, 2014, U.S. Patent 8 893 246.
[32] E. W. Bryksa and A. T. MacMillan, "Authorizing secured wireless access at hotspot having open wireless network and secure wireless network," Oct. 13, 2015, U.S. Patent 9 161 219.
[33] M. Al-Zubaidie, Z. Zhang, and J. Zhang, "RAMHU: A new robust lightweight scheme for mutual users authentication in healthcare applications," *Secur. Commun. Netw.*, vol. 2019, pp. 1–26, Mar. 2019.
[34] C. Matte, J. P. Achara, and M. Cunche, "Device-to-identity linking attack using targeted Wi-Fi geolocation spoofing," in *Proc. 8th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jun. 2015, pp. 1–6.
[35] S. D. Deshpande and T. J. Davenport, "Detection of rogue access point," Dec. 4, 2018, U.S. Patent 10 148 672.
[36] J. Harmon, "Systems and methods for detecting potentially illegitimate wireless access points," Jan. 30, 2018, U.S. Patent 9 882 931.
[37] P. Madani and N. Vlajic, "RSSI-based MAC-layer spoofing detection: Deep learning approach," *J. Cybersecur. Privacy*, vol. 1, no. 3, pp. 453–469, Aug. 2021.
[38] Y. Chen and J. Yang, "Defending against identity-based attacks in wireless networks," in *Handbook on Securing Cyber-Physical Critical Infrastructure*. Amsterdam, The Netherlands: Elsevier, 2012, pp. 191–222.
[39] H.-Y. Kim, "System and method for detecting rogue access point and user device and computer program for the same," Mar. 31, 2020, U.S. Patent 10 609 564.

[40] F. Breitinger, R. Tully-Doyle, and C. Hassenfeldt, "A survey on smartphone user's security choices, awareness and education," *Comput. Secur.*, vol. 88, Jan. 2020, Art. no. 101647.
[41] Z. Tang, Y. Zhao, L. Yang, S. Qi, D. Fang, X. Chen, X. Gong, and Z. Wang, "Exploiting wireless received signal strength indicators to detect evil-twin attacks in smart homes," *Mobile Inf. Syst.*, vol. 2017, Jan. 2017, Art. no. 1248578.



LUNODZO J. MWINUKA received the bachelor's degree in information technology and systems from Mzumbe University, Morogoro, Tanzania, in 2017. He is currently pursuing the master's degree in wireless and mobile computing with The Nelson Mandela African Institution of Science and Technology.

He joined Mzumbe University as an Academician in 2018, where he is currently teaching computing science studies at the Faculty of Science and Technology. His research interests include cyber-physical security, data privacy and control, intrusion detection, wireless security, and software coding security.

ABEL Z. AGGHEY received the degree in computer science and engineering from St. Joseph University, Dar es Salaam, Tanzania, in 2017. He is currently pursuing the master's degree in information systems and network security with The Nelson Mandela African Institution of Science and Technology.

In 2018, he joined the Department of Information Security and Communication Technology, Kamili Technologies Ltd., as a Security Specialist, where he is currently working. His research interests include artificial intelligence on cyber security, penetration testing/ethical hacking, intrusion detection and prevention systems, and data science.

SHUBI F. KAIJAGE (Senior Member, IEEE) currently works as an Associate Professor and the Dean of the School of Computational and Communications Science and Engineering (CoCSE), The Nelson Mandela African Institution of Science and Technology (NM-AIST), Arusha, Tanzania. He has published over 40 scientific articles in international peer-reviewed journals and more than 50 research papers presented in various international conferences. His research interests include optics and photonics, optical fiber and photonic crystal fibers (PCFs), fiber optics communication, terahertz wave technology, radio frequency identification (RFID), the Internet of Things (IoT), and wireless sensor networks.

He was a recipient of numerous international awards and grants.

JEMA D. NDIBWILE received the Engineering Doctorate degree in information security from the Nara Institute of Science and Technology, Japan, in 2019.

He is an Assistant Professor in cybersecurity at Carnegie Mellon University Africa. In assisting to address complex cyber security challenges, his specializations include cybersecurity, military intelligence, applied cryptography, ethical hacking, the psychology of cybersecurity, digital forensics, and cyber defenses. His current research interests include usable privacy and security, hacking countermeasures, the impact of artificial and human intelligence on cybersecurity, and social engineering approaches. He has extensive experience in ethical hacking/penetration testing, digital forensics, and project management leveraging tools, such as Kali Linux, Parrot OS, and Cellebrite.