WiFi Physical Layer Stays Awake and Responds When it Should Not

Ali Abedi, Haofan Lu, Alex Chen, Charlie Liu, and Omid Abari

***Abstract*—WiFi communication should be possible only between devices inside the same network. However, we find that all existing WiFi devices send back acknowledgments (ACK) to even fake packets received from unauthorized WiFi devices outside of their network. Moreover, we find that an unauthorized device can manipulate the power-saving mechanism of WiFi radios and keep them continuously awake by sending specific fake beacon frames to them. Our evaluation of over 5,000 devices from 186 vendors confirms that these are widespread issues. We believe these loopholes cannot be prevented, and hence they create privacy and security concerns. Finally, to show the importance of these issues and their consequences, we implement and demonstrate two attacks where an adversary performs battery drain and WiFi sensing attacks just using a tiny WiFi module which costs less than ten dollars.**

***Index Terms*—Mobile and Ubiquitous Systems, Security and Privacy, Smart Environment, Medium Access Control Protocols**

I. Introduciton

Today's WiFi networks use advanced authentication and encryption mechanisms (such as WPA3) to protect our privacy and security by stopping unauthorized devices from accessing our devices and data. Despite all these mechanisms, WiFi networks remain vulnerable to attacks mainly due to their physical layer behaviors and requirements defined by WiFi standards. In this paper, we find two loopholes in the IEEE 802.11 standard for the first time and show how they can put our privacy and security at risk.

a) WiFi radios respond when they should not. In a WiFi network, when a device sends a packet to another device, the receiving device sends an acknowledgment back to the transmitter. In particular, upon receiving a frame, the receiver calculates the cyclic redundancy check (CRC) of the packet in the physical layer to detect possible errors. If it passes CRC, then the receiver sends an Acknowledgment (ACK) to the transmitter to notify the correct reception of the frame. Surprisingly, we have found that all existing WiFi devices send back ACKs to even fake packets received from unauthorized WiFi devices outside of their network. Why should a WiFi device respond to a fake packet from an unauthorized device?!

b) WiFi radios stay awake when they should not. WiFi chipsets are mostly in sleep mode to save power. However, to make sure that they do not miss their incoming packets, they notify their WiFi access point before entering sleep mode so that the access point buffers any incoming packets for them. Then, WiFi devices wake up periodically to receive beacon frames sent by the associated access point. In regular operation, only the access point sends beacon frames to notify the devices that have buffered packets. When a device is notified, it stays awake to receive them. However, these beacon frames are not encrypted. Hence, we find that an unauthorized user can forge those beacon frames to keep a specific device awake for receiving the (non-existent) buffered frames.

We examine these behaviors and loopholes in detail over different WiFi chipsets from different vendors. Our examination of over 5,000 WiFi devices from 186 vendors shows that these are widespread issues. We then study the root cause of these issues and show that, unfortunately, they cannot be fixed by a simple solution such as updating WiFi chipsets firmware. Finally, we implement and demonstrate two attacks based on these loopholes. In the first attack, we show that by forcing WiFi devices to stay awake and continuously transmit, an adversary can continuously analyze the signal and extract personal information such as the breathing rate of the WiFi users. In the second attack, we show that by forcing WiFi devices to stay awake and continuously transmit, the adversary can quickly drain the battery, and hence disable WiFi devices such as home and office security sensors. These attacks can be performed from outside buildings despite the WiFi network and devices being completely secured. All the attacker needs is a $10 microcontroller with integrated WiFi (such as ESP32) and a battery bank. The attacker device can easily be carried in a pocket or hidden somewhere near the target building.

The main contributions of this work are:

- We find that WiFi devices respond to fake 802.11 frames with ACK, even when they are from unauthorized devices. We also find that WiFi radios can be kept awake by sending them fake beacon frames indicating they have packets waiting for them.
- We study these loopholes and their root causes in detail, and have tested more than 5,000 WiFi access points and client devices from more than 186 vendors.
- We implement two attacks based on these loopholes using just a 10-dollar off-the-shelf WiFi module and validate them in real-world settings.

II. Related Work

The loopholes we present in this paper are explored using packet injection, in which an attacker sends fake WiFi packets to devices in a secured WiFi network. Packet injection has

Ali Abedi is with the Department of Electrical Engineering, Stanford University, CA 94305, USA (email: abedi@stanford.edu)

Alex Chen and Charlie Liu are with the School of Computer Science, University of Waterloo, ON N2L 3G1, Canada (email: zihanchen.ca@gmail.com, charlie.liu@uwaterloo.ca)

Omid Abari and Haofan Lu are with the Department of Computer Science, University of California at Los Angeles, CA 90095, USA (email: omid@cs.ucla.edu, haofan@cs.ucla.edu)

been used in the past to perform various types of attacks against WiFi networks such as denial of service attacks for a particular client device or total disruption of the network [1], [2], [3], [4]. These attacks use different approaches such as beacon stuffing to send false information to WiFi devices [5], [6], or Traffic Indication Map (TIM) forgery to prevent clients from receiving data [7], [8]. However, all of these attacks focus on spoofing 802.11 MAC-layer management frames to interrupt the normal operation of WiFi networks. To provide a countermeasure for some of these attacks, the 802.11w standard [9] introduces a protected management frame that prevents attackers from spoofing 802.11 management frames. Instead of spoofing 802.11 MAC frames, we exploit properties of the 802.11 physical layer to force a device to stay awake and respond when it should not. These loopholes open the door to multiple research avenues including new security and privacy threats.

WiFi sensing attack: Over the past decade, there has been a significant amount of research on WiFi sensing where WiFi signals are used to detect human activities [10], [11], [12], [13], [14], [15], [16], [17], [18]. However, these systems target applications with social benefits and cannot be easily used by an attacker to create privacy and security threats. This is because either these techniques require cooperation from the target WiFi device or the attacker needs to be very close to the target to use these systems. A recent study shows that by capturing WiFi signals coming out of a private building, it is possible for an adversary to track user movements inside that building [19]. However, this attack has a bootstrapping stage which requires the attacker to walk around the target building for a long time to find the location of the WiFi devices. Furthermore, since this work relies on only the normal intermittent WiFi activities, it cannot capture continuous data such as breathing rate.

Battery draining attack: Battery draining attacks date back to 1999 [20] and there have been many studies on such attacks and potential defense mechanisms since then [21]. Battery discharge models and energy vulnerability due to operating systems have been investigated [22], [23]. A more recent study plays multimedia files implicitly to increase power consumption during web browsing [24], [25]. In terms of defending, a monitoring agent that searches for abnormal current draw is discussed in [26]. In contrast, our attack exploits the loopholes in the 802.11 physical layer protocol and the power-hungry WiFi transmission to quickly drain a target device's battery. We will discuss in Section III-B that stopping our proposed attack is nearly impossible on today's WiFi devices.

This paper is an extension of our previous workshop publication [27]. The workshop paper shows preliminary results for our finding that WiFi devices respond with ACKs to packets received from outside of their network, and provides a brief discussion on potential privacy and security concerns of this behavior without studying them. We have also explored how the WiFi power saving mechanism can be exploited to keep a target device awake in a localization attack [28]. In this paper, we provide an in-depth study of these previously discovered loopholes. We also design and perform two privacy and security attacks, based on these loopholes. Finally, we implement these attacks on off-the-shelve WiFi devices and present detailed performance evaluations.

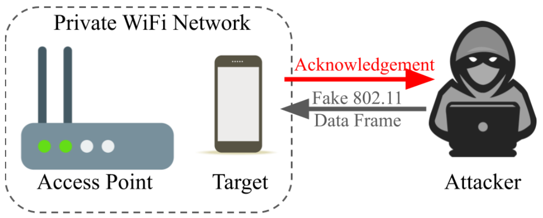
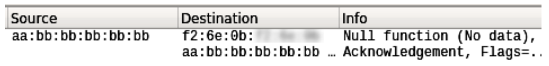


Fig. 1: WiFi devices send an ACK for any frame they receive without checking if the frame is valid.



```
Source              Destination              Info
aa:bb:bb:bb:bb:bb   f2:6e:0b:[redacted]      Null function (No data),
                    aa:bb:bb:bb:bb:bb …      Acknowledgement, Flags=..
```

Fig. 2: Frames exchanged between attacker and victim in WiFi responds when it should not attack.

III. WiFi Responds When It Should Not

Most networks use security protocols to prevent unauthorized devices from communicating with their devices. Therefore, one may assume that a WiFi device only acknowledges frames received from the associated access point or other devices in the same network. However, we have found that all today's WiFi devices acknowledge even the frames they receive from an unauthorized device from outside of their network. In particular, as long as the destination address matches their MAC address, their physical layer acknowledges it, even if the frame has no valid payload. In this section, we examine this behavior in more detail, and explain why this problem happens and why it is not preventable.

To better understand this behavior, we run an experiment where we use two WiFi devices to act as a victim and an attacker. The attacker sends fake WiFi packets to the victim. We monitor the real traffic between the attacker and the victim's device.

Setup: For the victim, we use a tablet, and for the attacker, we use a USB WiFi dongle that has a Realtek RTL8812AU 802.11ac chipset. This is a $12 commodity WiFi device. The attacker uses this device to send fake frames to the victim's device. To do so, we develop a python program that uses the Scapy library [29] to create fake frames. Scapy is a python-based framework that can generate arbitrary frames with custom data in the header fields. Note, that the only valid information in the frame is the destination MAC address (i.e., the victim's MAC address). The transmitter MAC address is set to a fake MAC address (i.e., aa:bb:bb:bb:bb:bb), and the frame has no payload (i.e., null frame) and is not encrypted.

Result: Figure 2 shows the real traffic between the attacker and the victim device captured using Wireshark packet sniffer [30]. As can be seen, when the attacker sends a fake frame to the victim, the victim sends back an ACK to the fake MAC address (aa:bb:bb:bb:bb:bb). This experiment confirms that WiFi devices acknowledge frames without checking their validity. Finally, to see if this behavior exists on other WiFi devices, we have repeated this test with a variety of devices (such as

Device	WiFi module	Standard
MSI GE62 laptop	Intel AC 3160	11ac
Ecobee3 thermostat	Atheros	11n
Surface Pro 2017	Marvel 88W8897	11ac
Samsung Galaxy S8	Murata KM5D18098	11ac
Google Wifi AP	Qualcomm IPQ 4019	11ac

TABLE I: List of tested chipsets/devices

WiFi Client Device		WiFi Access Point	
Vendor	# devices	Vendor	# devices
Apple	143	Hitron	723
Google	102	Sagemcom	601
Intel	66	Technicolor	410
Hitron	65	eero	195
HP	63	Extreme N.	188
Samsung	56	Cisco	156
Espressif	47	HP	104
Hon Hai	46	TP-LINK	101
Amazon	41	Google	80
Sagemcom	38	D-Link	75
Liteon	33	NETGEAR	69
AzureWave	30	ASUSTek	51
Sonos	30	Aruba	46
Nest Labs	27	SmartRG,	44
Murata	24	Ubiquiti N.	35
Belkin	20	Zebra	35
TP-LINK	20	Pegatron	28
Cisco	16	Belkin	25
ecobee	13	Mitsumi	25
Microsoft	13	Apple	19
Others	630	Others	789
Total	1523	Total	3805

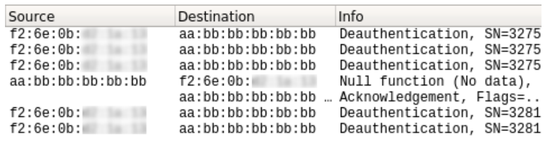
TABLE II: List of WiFi devices and APs that respond to our fake 802.11 frames.

laptops, smart thermostats, tablets, smartphones, and access points) with different WiFi chipsets from different vendors, as shown in Table I. Note, target devices are connected to a private network and the attacker does not have their secret key. After performing the same experiment as before, we found that all of these devices also respond to fake packets received from a device outside of their network.

A. How widespread is this loophole?

In the previous section, we examined a few different WiFi devices and showed that they are all responding to fake frames from unauthorized devices. Here, we examine thousands of devices to see how widespread this behavior is. In the following, we explain the setup and results of this experiment.

Setup: To examine thousands of devices, we mounted a WiFi dongle on the roof of a vehicle and drove around the city to test all nearby devices. For the WiFi dongle, we use the same Realtek RTL8812AU USB WiFi dongle, and connect it to a Microsoft Surface, running Ubuntu 18.04. We develop a multi-threaded program using the Scapy library [29] to discover nearby devices, send fake 802.11 frames to the discovered devices, and verify that target devices respond to our fake frames. Specifically, our implementation contains three threads. The first thread discovers nearby devices by sniffing WiFi traffic and adding the MAC address of unseen devices to a target list. The second thread sends fake 802.11 frames to the list of target devices. Finally, the third thread checks to verify that target devices respond with an ACK.



```
Source                Destination            Info
f2:6e:0b:[redacted]   aa:bb:bb:bb:bb:bb      Deauthentication, SN=3275
f2:6e:0b:[redacted]   aa:bb:bb:bb:bb:bb      Deauthentication, SN=3275
f2:6e:0b:[redacted]   aa:bb:bb:bb:bb:bb      Deauthentication, SN=3275
aa:bb:bb:bb:bb:bb     f2:6e:0b:[redacted]    Null function (No data),
                      aa:bb:bb:bb:bb:bb …    Acknowledgement, Flags=..
f2:6e:0b:[redacted]   aa:bb:bb:bb:bb:bb      Deauthentication, SN=3281
f2:6e:0b:[redacted]   aa:bb:bb:bb:bb:bb      Deauthentication, SN=3281
```

Fig. 3: The attacked access point detects that something strange is happening, however it still ACKs fake frames

Results: We perform this experiment for one hour while driving around the city. In total, we discovered 5,328 WiFi nodes from 186 vendors. The list includes 1,523 different WiFi client devices from 147 vendors and 3,805 access points from 94 vendors. Table II shows the top 20 vendors for WiFi devices and WiFi access points in terms of the number of devices discovered in our experiment. The list includes devices from major smartphone manufacturers (such as Apple, Google, and Samsung) and major IoT vendors (such as Nest, Google, Amazon, and Ecobee). We found that all 5,328 WiFi Access Points and devices responded to our fake 802.11 frames with an acknowledgment, and hence we infer that most probably all of today's WiFi devices and access points respond to fake frames when they should not.

B. What is the root cause of this loophole?

So far, we have demonstrated that all existing WiFi devices respond to fake packets received from unauthorized WiFi devices outside of their network. Now, the next question is why this behavior exists, and if it can be prevented in future WiFi chipsets.

In a WiFi device, when the physical layer receives a frame, it checks the correctness of the frame using error-checking mechanisms (such as CRC) and transmits an ACK if the frame has no error. However, checking the validity of the content of a frame is performed by the MAC and higher layers. Unfortunately, this separation of responsibilities and the fact that the physical layer does not coordinate with higher layers about sending ACKs seem to be the root cause of the behavior. In particular, we have observed that when some access points receive fake frames, they start sending *deauthentication frames* to the attacker, requesting it to leave the network. These access points detect the attacker as a "malfunctioning" device and that is why they send deauthentication frames. Surprisingly, although the access points have detected that they are receiving fake frames from a "malfunctioning" device, we found that they still acknowledge the fake frames.

An example traffic that demonstrates this behavior is shown in Figure 3. As can be seen, although the access point has already sent three deauthentication frames to the attacker, it still acknowledges the attacker's fake frame. We then manually blocked the attacker's fake MAC address on the access point. Surprisingly, we observed that the AP still acknowledges the fake frames. These observations verify that sending ACK frames happens automatically in the physical layer without any communication with higher layers. Therefore, the software running on the access points does not prevent the physical layer from sending ACKs to fake frames.

IV. WiFi Stays Awake When It Should Not

We have also found a loophole that allows an unauthorized device to keep a WiFi device awake all the time. One may think that a WiFi device can be kept awake by just sending fake back-to-back packets to it and forcing it to transmit acknowledgment. However, this approach does not work. Most WiFi radios go to sleep mode to save energy during inactive states such as screen lock, during which the attacker is not able to keep them awake by sending back-to-back packets. However, we have found a loophole in the power saving mechanism of WiFi devices which can be used by an unauthorized device to keep any WiFi device awake all the time.

A. How does WiFi power saving mechanism work?

Wireless tranceivers are very power-hungry. Therefore, WiFi radios spend most of the time in the sleep mode to save power. When a WiFi radio is in sleep mode, it cannot send or receive WiFi packets. To avoid missing any incoming packets, when a WiFi device wants to enter the sleep mode it notifies the WiFi access point so that the access point buffers any incoming packets for this device. WiFi devices, however, wake up periodically to receive beacon frames to find out if packets are waiting for them. In particular, WiFi access points broadcast beacon frames periodically which includes a Traffic Indication Map (TIM) field that indicates which devices have buffered packets on the access point. For example, if the association ID of a WiFi device is x, then the $(x+1)^{th}$ bit of TIM is assigned to that device. Finally, when a device is notified that has some buffered packets on the access point, it stays awake and replies with a *Null-function* packet with a power management bit set to ”0”. In this way, the WiFi device informs the access point it is awake and ready to receive packets.

B. How can one manipulate power saving?

We have found that an unauthorized device can use the power-saving mechanism of WiFi devices to force them to stay awake. In particular, an attacker can pretend to be the access point and broadcasts fake beacon frames indicating that the AP has buffered traffic toward WiFi devices, forcing the WiFi devices to stay awake as illustrated in Figure 4. However, this requires the attacker to know the MAC address and the SSID of the network's access point, as well as the association ID and MAC address of the targeted device so that it can set the correct bit in TIM. The access point MAC address and SSID can be easily discovered by sniffing the WiFi traffic using software such as Wireshark since the MAC address is never encrypted and all nodes send packets to the access point. To keep a WiFi device awake, the attacker pretends to be the access point by transmitting fake beacon frames with the TIM field set to 0xFF, indicating all client devices have buffered traffic. By setting the TIM field to 0xFF, the attacker does not need to know the association ID of the target device because we claim that everyone should wake up. To avoid keeping all WiFi devices awake, we find that one can send a fake beacon frame as a unicast packet, instead of the usual broadcast beacons. This way only the target device receives the packet and we do not interfere with the operation of other devices. Interestingly, our experiments show that devices do not care if they receive beacons as broadcast or unicast frames.

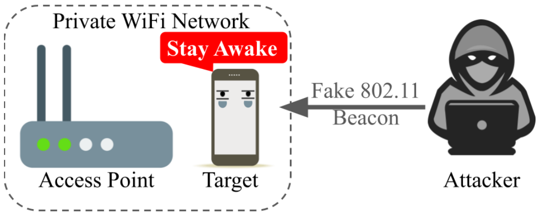
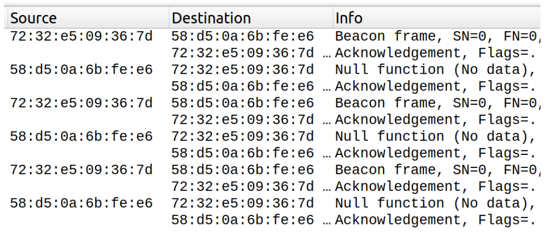


Fig. 4: WiFi devices stay awake on hearing a forged beacon frame with TIM flags set up.



```
Source             Destination          Info
72:32:e5:09:36:7d  58:d5:0a:6b:fe:e6    Beacon frame, SN=0, FN=0,
                   72:32:e5:09:36:7d …  Acknowledgement, Flags=.
58:d5:0a:6b:fe:e6  72:32:e5:09:36:7d    Null function (No data),
                   58:d5:0a:6b:fe:e6 …  Acknowledgement, Flags=.
72:32:e5:09:36:7d  58:d5:0a:6b:fe:e6    Beacon frame, SN=0, FN=0,
                   72:32:e5:09:36:7d …  Acknowledgement, Flags=.
58:d5:0a:6b:fe:e6  72:32:e5:09:36:7d    Null function (No data),
                   58:d5:0a:6b:fe:e6 …  Acknowledgement, Flags=.
72:32:e5:09:36:7d  58:d5:0a:6b:fe:e6    Beacon frame, SN=0, FN=0,
                   72:32:e5:09:36:7d …  Acknowledgement, Flags=.
58:d5:0a:6b:fe:e6  72:32:e5:09:36:7d    Null function (No data),
                   58:d5:0a:6b:fe:e6 …  Acknowledgement, Flags=.
```

Fig. 5: Frames exchanged between attacker and victim in stay away attack

In Figure 5, we show some example traffic during this attack. The first packet in the figure is a fake beacon frame with a spoofed MAC address of an AP, with the actual MAC address of the target device. Because this is a unicast packet the target device ACKs the frame (i.e., second packet in the figure). Because the beacon frame claims that there are packets waiting for the target on the AP, the target device stays awake and sends a null packet to the AP indicating that it is ready to receive the packets (i.e., third packet in the figure). Finally the actual AP ACKs this packet indicated as the forth packet in Figure 5. At this point, nothing else happens because the AP does not actually have any packet for the target. However, the attacker keeps sending these fake beacons to the target to keeps the target device awake. In Figure 5, we also show two more rounds of such packet exchanges. As long as this process continues, the target device stays awake, and does not go back to sleep.

V. Privacy Implication: WiFi Sensing Attack

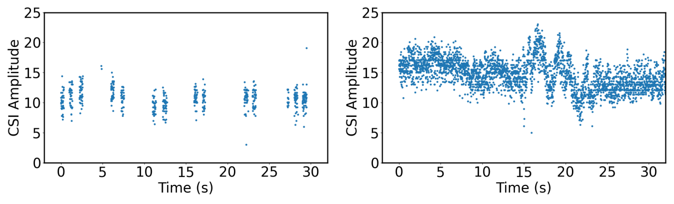
Recently, there has been a significant amount of work on WiFi sensing technologies that use WiFi signals to detect events such as motion, gesture, and breathing rate. In this section, we show how an adversary can combine WiFi sensing techniques with the above loopholes to monitor people's breathing rate whenever she/he wants from outside buildings despite the WiFi network and devices being completely secured. In particular, an adversary can force our WiFi devices to stay awake and continuously transmit WiFi signals. Then she/he can continuously analyze our signals and extract information such as our breathing rate. Note, since most of the time, we are close to a WiFi device (such as a smartwatch, laptop, or tablet), our body will change the amplitude and phase of the signals which can be easily extracted by the adversary. Here we present an experiment that demonstrates how the two loophole we described before can be used to estimate someone's breathing rate from a distance.

Setup: Similar to the experiment described in Section III, we use an RTL8812AU USB dongle to inject fake packets to a smartphone held by a person who is watching YouTube on the phone. The distance between the smartphone and the user is about 60 cm. The attacking device and the victim are several meters apart in two separate rooms with no line of sight between them. The attacker also uses an ESP32 WiFi module to record the Channel State Information (CSI) of received ACKs. We use the ESP32's WiFi Driver API [31] to collect CSI. We put ESP32 into the promiscuous mode and register a customized callback function using the esp_wifi_set_csi_rx_cb function. The callback function is invoke every time a packet is received at the physical layer. The CSI data is sent to the callback function as a C struct. According to the documentation, the CSI is represented an array of 8-bit signed integers. For generality, we use the CSI measurement of legacy long training field (LLTF), which contains 128 bytes, corresponding to the real and imaginary parts of the 64 subcarriers, respectively. Note that the ACK frames received from the target device is transmitted using legacy 802.11 bitrates and only their LLTF can be used to measure the CSI. Although all 64 subcarriers are reported, only the 52 data subcarrier have valid CSI values. For more details, we refer readers to the official documentation [31]. We use the monitor command of ESP-IDF [32] to retrieve CSI data on a laptop. For each packet, we record the system timestamp at which the packet is received, as well as the CSI measurement. We save the records to a file for further processing.

In the WiFi promiscuous mode, every packet received by ESP32 triggers the callback function to extract CSI, which significantly slows down the CSI extraction process which causes buffer overflow. To solve this problem, we apply multiple packet filters to screen out the unwanted packets. Firstly, we use the internal esp_wifi_set_promiscuous_filter API to remove Data frames and Management frames. Among control frames, we further use the esp_wifi_set_promiscuous_ctrl_filter API to get only the ACK frames. Finally, we filter the packets using the receiver's MAC address to get rid of packets that were not transmitted by the target device. Our raw CSI traces collected during our experiments are made available online at [33].

Result: We run two experiments using this setup. First we only send fake 802.11 packets to the target device. Figure 6a plots the amplitude of CSI over time for the ACK packets received from the target device. As can be seen, the responses are sparse and discontinued even when the attacker sends back-to-back packets to the WiFi device. This is because the WiFi device goes to sleep mode frequently. Therefore, the first loophole is not enough to extract breathing rate information.

In the second experiment, we introduce fake beacon frames (i.e., the second loophole) to keep the target device awake some that we can collect CSI samples continuously. We find that although sending fake beacon frames keeps the target device awake, sending them very frequently will cause WiFi devices to recognize the suspicious attacker's behavior and disconnect from it. Therefore, to keep the WiFi device awake, instead of just sending beacon frames back-to-back, the attacker can continuously transmit normal fake packets to a WiFi device and periodically sends fake beacon frames to keep it awake. Figure 6b shows the result of an experiment where the attacker is continuously transmitting fake packets to a WiFi device and periodically sends fake beacon frames (i.e., every 5th packet). As can be seen, the target device is continuously awake and responding to fake packets with ACKs. In the rest of this section, we describe different technical challenges in executing this attack and our solutions, along with performance evaluation in a variety of environments.



(a) Without fake beacon frames (b) With fake beacon frames

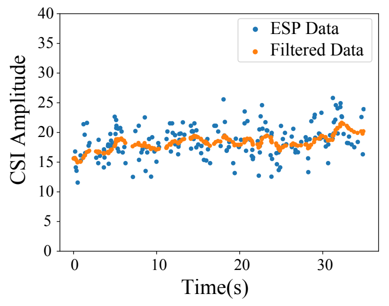
Fig. 6: CSI amplitude of ACKs responded by the target device

A. Attack Design, Scenarios and Setup

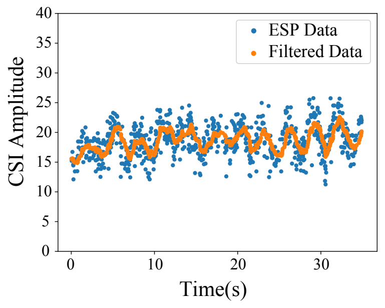
1) Attack Design: The attacker sends fake packets to a WiFi device in the target property and pushes it to transmit ACK packets. In particular, since an adult's normal breathing rate is around 12 -20 times per minute (i.e., 0.2- 0.33Hz), receiving several ACK packets per second is sufficient for the attacker to estimate the breathing rate, without impacting the performance of the target WiFi network. The attacker then takes the Fourier transform of the CSI information of ACK packets to estimate the breathing rate of the person who is nearby the WiFi device. However, due to the random delays of the WiFi random access protocol and the operating system's scheduling protocol, the collected data samples are not uniformly spaced in time. Hence, the attacker cannot simply use standard FFT to estimate the breathing rate. Instead, they need to use a non-uniform Fourier transform, and a voting algorithm [19], [34], [35], [36] to extract the breathing rate. The Non-Uniform Fast Fourier Transform (NUFFT) algorithm 1 used is shown below.

```
Algorithm 1: Non-uniform FFT
Data: Time indices $t$, data samples $x$ of length $n$
Result: Magnitude of each frequency component
$d \leftarrow \min_i(t_i - t_{i-1}) \quad i = 1, 2, ..., n.$;
for $i \leftarrow 1$ to $n-1$ do
    $interval \leftarrow t[i] - t[i-1]$;
    if $interval > d$ then
        $count \leftarrow \lfloor \frac{interval}{d} \rfloor$;
        Interpolation($t$, $x$, $t[i]$, $t[i-1]$, $count$);
    end
end
return FFT($t$, $x$)
```

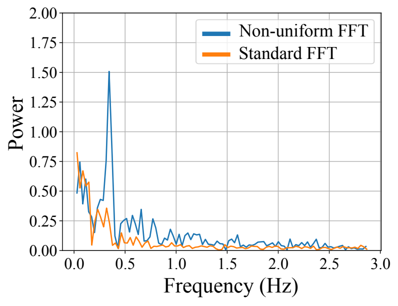
The algorithm first finds the minimum time gap between any two adjacent data points d, then linearly interpolates any



(a) Before interpolation



(b) After interpolation



(c) Standard FFT vs non-uniform FFT

Fig. 7: Steps to extract breathing rate from the CSI.

interval that is larger than the gap with $\lfloor interval/d \rfloor$ samples. Finally, it uses a regular FFT algorithm to find the magnitude of each frequency component. A low-pass filter is applied before feeding data to the FFT analysis to reduce noise (not shown in the algorithm).

Figure 7(a) and 7(b) show the amplitude of CSI before and after interpolation, respectively, when the attacker sends 10 packets per second to a WiFi device that is close to the victim. Each figure shows both the original data (in blue) and the filtered data (in orange). Figure 7(c) shows the frequency spectrum of the same signals when a standard FFT or our non-uniform FFT is applied. A prominent peak at 0.3Hz is shown in the non-uniform FFT spectrum, indicating a breathing rate of 18 bpm.

WiFi CSI gives us the amplitude of 52 subcarriers per packet. We observed that these subcarriers are not equally sensitive to the motion of the chest. Besides, a subcarrier's sensitivity may vary depending on the surrounding environment due to frequency selective fading. For a more reliable attack, the attacker should identify the most sensitive subcarriers over a sampling window. Previously proposed voting mechanisms for coarse-grained motion detection applications [19], [34], [35], [36] cannot be directly applied in this situation, as chest motion during respiration is at a much smaller scale. Instead, we developed a soft voting mechanism, where each subcarrier gives a weighted vote to a breathing rate value. Soft voting refers to the fact that each subcarrier associates a weight or probability to its vote and the final aggregated decision (that considers all subcarriers' votes) may differ from an individual subcarrier's vote. The breathing rate that gets the most votes is reported. Specifically, We first find the power of the highest peak (P_{peak}) in the FFT, and then calculate the average power of the rest of the bins (P_{ave}). The exponent of the Peak-to-Average Ratio (PAR): $e^{\frac{f_{peak}}{f_{ave}}}$ is used as the weight of the corresponding subcarrier. In this way, we guarantee the subcarriers with higher SNR have significantly more votes than the rest of the subcarriers.

2) Attack Scenarios: We evaluate the WiFi sensing attack in different scenarios, both indoor and outdoor. In the indoor scenario, the attacker and the target are placed in the same building but on different floors. The height of one floor in the building is around 2.8 m. This scenario is similar to when the attacker and the target person are in different units of an apartment or townhouse. In the outdoor scenario, the attacker is outside the target's house. For the outdoor experiments, the attacker is in another building which is around 20 m away from the target building. In all of the experiments, the target WiFi devices are placed 0.5 to 1.4 m away from the target person's body. The person is either watching a movie, typing on a laptop, or surfing the web using his cell phone. During the experiments, other people are walking or busy with other daily activities in the house. Finally, we run the attack and compare the estimated breathing rate with the ground truth. To obtain the ground truth, we record the target person's breathing sound by attaching a microphone near his/her mouth [37]. We then calculate the FFT on the sound signal to measure the breathing frequency. Note that the attack does not need this information and this is just to obtain the ground truth in our experiments.

3) Attacker Setup: **Hardware Setup:** The attacker uses a Linksys AE6000 WiFi card and an ESP32 WiFi module [38] as the attacking device. Both devices are connected to a ThinkPad laptop via USB. The Linksys AE6000 is used to send fake packets and the ESP32 WiFi module is used to receive acknowledgments (ACK) and extract CSI. Although we use two different devices for sending and receiving, one can simply use an ESP32 WiFi module for both purposes. The use of two separate modules gave us more flexibility in running many experiments. As for the target device, we use a One Plus 8T smartphone without any software or hardware modifications. We have also tested our attack on an unmodified Lenovo laptop, a Microsoft Surface Pro 4 laptop, and a USB WiFi card as the target device and we obtained similar results. It is worth mentioning that any WiFi device can be a target without any software or hardware modification.

Software Setup: We implemented the fake packet injection algorithm using Scapy [29], the CSI collection script using ESP-IDF [31], and breathing rate estimation algorithm in Python on a laptop. We inject 30 fake Null function frames to the target device per second, mixed with 3 fake beacon frames to keep the device awake. The collected CSI data is fed to the algorithm which produces the breathing rate estimation values in real-time. To process this data in real time, a sliding window (buffer) is used. The size of the window is 30 s and the stride step is 1 s. 30 seconds is a large enough window for estimating a stable breathing rate value. Note that an adult breathes around 6 times during such a window. The window is a queue of data points, and it updates every second by including 1 second of new data points to its head

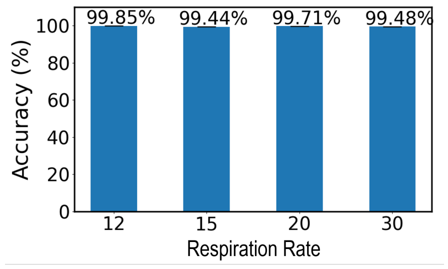


Fig. 8: The average accuracy of the attack in estimating the target person's breathing rate when attacker and target device are in the same building.

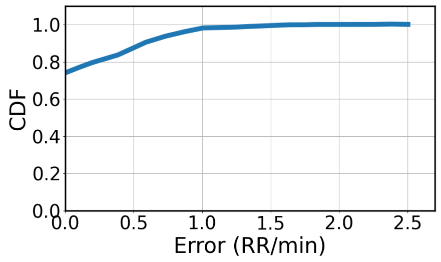


Fig. 9: The CDF of the error in estimating the Respiration Rate (RR) when attacker and target device are on different floors of the same buildin.

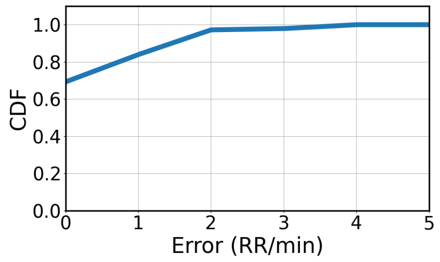


Fig. 10: The CDF of the error in estimating the Respiration Rate (RR) when he attacker and target device are in different buildings (20m away)

and removing 1 second of old data points from its tail. The breathing rate estimation runs the analysis algorithm on the data points inside the window whenever it is updated. The window slides once per second. Hence, our software reports an estimation of breathing rate every second. Note that there is a 30-second delay at the beginning since the window needs to be filled first.

B. Results

We evaluate the effectiveness of the attack in different scenarios such as when the attacker and the target are in the same building or different buildings.

1) Accuracy in Detecting Breathing Rate: **Same Building Scenario:** First, we evaluate the accuracy of the attack by estimating the breathing rate in an indoor scenario where the target device and attacker are in the same building. We evaluate the accuracy when the target's breathing rate is 12, 15, 20, and 30 breaths per minute. Note, that the normal breathing rate for an adult is 12-20 breaths per minute while resting, and higher when exercising. In this experiment, the user is watching a video. To make sure the target person's breathing rate is close to our desired numbers, we place a timer in front of the person, where they can adjust their breathing rate accordingly. This is just to better control the breathing rate during the experiment and is not a requirement nor an assumption in this attack. We run each experiment for two minutes and we collect about 70 CSI samples per second adding up to about 8400 samples. During this time, we collect the estimated breathing rate from both ground truth and the attack for different locations of the target device. Figure 8 shows the average accuracy in estimating breathing rate across all experiments. The accuracy is calculated as the ratio of the estimated breathing rate reported by the attack over the ground truth breathing rate. The figure shows that the accuracy of estimating the breathing rate is over 99% in all scenarios. Finally, Figure 9 plots the Cumulative Distribution Function (CDF) of the error in detecting Respiration Rate (RR) for over 2400 measurements. The figure shows that 78% of the estimated results have no error. The figure also shows that 99% of measurements have less than one breath per minute error which is negligible.

Different Building Scenario: So far, we have evaluated our attack where the target and the attacker are in different rooms or floors of the same building. Here we push this further and examine whether our attack works if the attacker and the target person are in a different building. We place the target device in a building on a university campus on a weekday with people around. A person is sitting around 0.5 m away from the device. We then place the attacker in another building which is around 20 m away from the target building. We use an external patch antenna with 3 dBi gain for this experiment. We ran this experiment for 3 minutes, with a sampling rate of about 70 samples per second. In total, we collected 12600 samples. Similar to the previous experiment, we run the attack and compare the estimated breathing rate with the ground truth. Figure 10 shows the CDF of error for 180 measurements in this experiment. Our results show that the attacker successfully estimates the breathing rate. Note, that the reason that the attack works even in such a challenging scenario with other people being around is two-fold. First, using an FFT helps to filter out the effect of most non-periodic movements and focuses on periodic movements and patterns. Second, wireless channels are more sensitive to changes as we get closer to the transmitter [39], and since in these scenarios, the target person is very close to the target device, their breathing motion has a higher impact on the CSI signal compared to the other movements in the environment.

2) Human Presence Detection: We next evaluate the efficacy of detecting whether there is a target person near the WiFi device or not. In this experiment, the target phone is placed on a desk and the person stays around the device for 30 seconds, then walks away from the device, and then comes back near the device. Note, in our algorithm, when there is no majority vote during the voting phase, we return -1 to indicate no breathing detected. Figure 11 shows the results of this experiment. As illustrated in the figure, we can correctly detect the breathing rate when a person is near the device. In other words, the algorithm can detect if there is no one near the target device and refrain from reporting a random value.

3) Effect of Distance and Orientation: Next, we evaluate the effectiveness of the attack for different orientations of the device with respect to the person. We also evaluate its performance for different distances between the target device and the target person.

Orientation: We evaluate the effect of orientation of the target person with respect to the target device (laptop). We run the

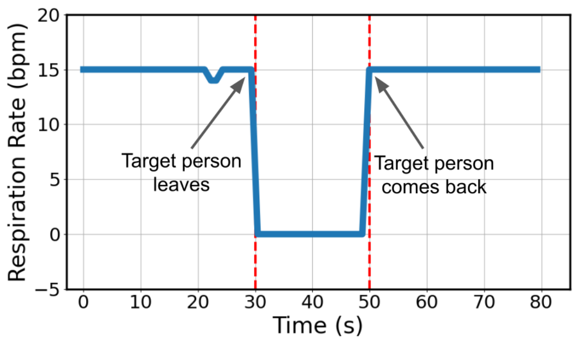


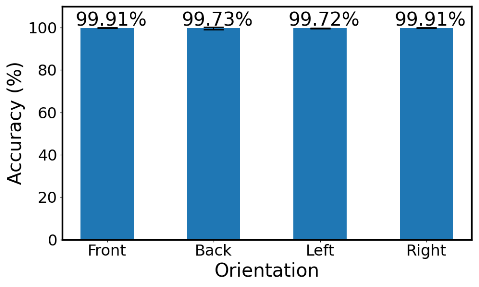
Fig. 11: The efficacy of estimating the breathing rate when there is no target near the WiFi device.

same attack as before for different orientations (i.e. sitting in front, back, left, and right side of a laptop). The user is 0.5m away from the target device in all cases. We ran each experiment for 2 minutes, with a sampling rate of 50 samples per second. Figure 12a shows the result of this experiment. Each bar shows the average accuracy for 90 measurements. Our result shows that regardless of the orientation of the person with respect to the device, the attack is effective and detects the breathing rate of the person accurately. In particular, even when the person was behind the target device, the attack still detects the breathing rate with 99% accuracy.

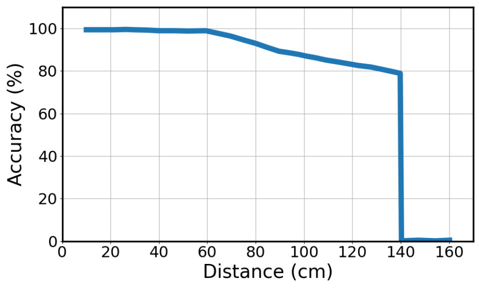
Distance: Here, we are interested to find out what the maximum distance between the target device and the person can be while the attacker still detects the person's breathing rate. To do so, we place the attacker device and the target device 5 meters apart in two different rooms with a wall in between. We then run different experiments in which the target person stays at different distances from the target device. In each experiment, we measure the breathing rate for two minutes and calculate the average breathing rate over this time. Finally, we compare the estimated breathing rate to the ground truth and calculate the accuracy as mentioned before.

Figure 12b shows the results of this experiment. The accuracy is over 99% when the distance between the target device and the target person is less than 60 cm. Note, in reality, people have their laptops or cellphone very close to themselves most of the time, and 60 cm is representative of these situations. The accuracy drops as we increase the distance. However, even when the device is at 1.4 m from the person's body, the attack can still estimate the breathing rate with 80% accuracy. Note, this is the accuracy in finding the absolute breathing rate and the change in the breathing rate can be detected with much higher accuracy. Finally, the figure shows that the accuracy suddenly drops to zero for a distance beyond 1.4 m. This is due to the fact that at that distance the power of the peak at the output of the FFT goes below the noise floor, and hence, the peak is not detectable.

4) Effect of Multiple People: Last, we evaluate if the attack can be used to detect the breathing rate of multiple people simultaneously. We test our attack in three different scenarios. In the first scenario, two people are near the laptop while one is working on the laptop and the other is just sitting next to him, as shown in Figure 13a. The attacker targets the



(a) various orientations



(b) different distances.

Fig. 12: effectiveness of the attack for different orientation and distance of the targeted WiFi device respect to the person.

laptop and tries to estimate their breathing rate. Note, that the attacker has no prior information about how many people are next to the laptop. In the second scenario, we repeat the same experiment as the first scenario except that the second person is sitting behind the laptop, as shown in Figure 13b. In the third scenario, there are two people in the same space but each person is next to a different device. The attacker targets the laptops and tries to estimate their breathing rates. In these experiments, the target device is 0.5-0.7 m away from the person. We ran each experiment for 1 minute with a sampling rate of 50 samples per second.

Figure 13c shows the results for this evaluation. The blue bars show the result for the first person who is working on the laptop, and the red bars show the results for the second person. Our results show that the attack effectively detects the breathing rate of both people regardless of their orientation. However, the accuracy in detecting the breathing rate for the second person is a bit lower than the first person for the first and second scenarios. This is because the second person's distance to the target device is slightly more and hence the accuracy has decreased.

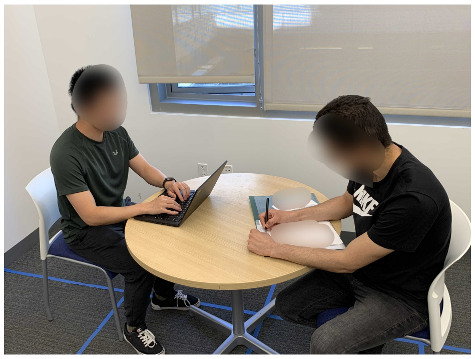
VI. Security Implication: Battery Drain Attack

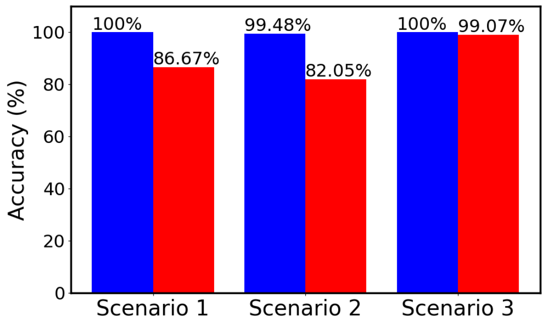
In this section, we show how an adversary can drain the battery of our WiFi devices by using the above loopholes and forcing our WiFi devices to stay awake and continuously transmit WiFi signals.

A. Attack Design and Setup

1) Attack Design: The attacker forces the target device to stay awake and continuously transmit WiFi packets by sending it back-to-back fake frames and some periodic fake beacons. However, to maximize the amount of time the target device

(a) Scenario 1


(b) Scenario 2



(c) Breathing Rate Estimation of two persons

Fig. 13: Accuracy under three different scenarios: Scenario 1: two people sit side-by-side in front of the target device; Scenario 2: one person sits in front of the target device, the other one sits behind the target device; Scenario 3: two people sit in front of two target devices, respectively. Attacker attacks one by one.

spends transmitting, we study a few different types of fake query packets that the attacker can send. Note, that the power consumption of transmission is typically higher than that of reception.[1] Hence, to maximize the battery drain, we want to send a short query packet and receive a long response.

Query	Query size	Response	Response size
Null	28 bytes	ACK	14 bytes
RTS	20 bytes	CTS	14 bytes
BAR	24 bytes	BA	32 bytes

TABLE III: Different types of fake queries and their responses. Note, Null is a data packet without any payload. BAR and BA stand for Block ACK Request, and Block ACK, respectivly.

Table III lists some query packets and their corresponding responses. The best choice for a query packet is Block ACK requests since the target will respond with a Block ACK that is larger than other query responses. Another important factor to consider for maximizing the battery drain is the bitrate. When the bitrate of the query packet increases, the bitrate of the response will also increase as specified in the IEEE 802.11 standard. Hence, at first glance, it may seem that to maximize the battery drain, the attacker must use the fastest bitrate possible to transmit query packets, forcing the target device to transmit as many responses as possible. However, it turns out that this is not the case. The power consumption depends mostly on the amount of time the target device spends transmitting packets. Hence, when a higher rate is used for the query and response packets, the total time the target spends on transmission does not increase. In fact, the total time spent transmitting decreases mainly due to overheads such as channel sensing and backoffs. For example, if we increase the bitrate by 6 times (i.e., from 1 Mbps to 6 Mbps), the number of packets will increase by only 3.3 times. As a result, to maximize the transmission time of the target device, the attacker should use the lowest rate for the query packet.

2) Attack Setup: **Attacking device:** Any WiFi card capable of packet injection can be used as the attacking device. We use a USB WiFi card connected to a laptop running Ubuntu 20.04. The WiFi card has an RTL8812AU chipset [41] that supports IEEE 802.11 a/b/g/n/ac standards. We have installed the aircrack-ng/rtl8812au driver [42] for this card which enables robust packet injection. We utilize the Scapy [29] library to inject fake WiFi packets to the target device. Scapy allows defining customized packets and multiple options for packet injection. Since we need to inject many packets in this attack, we use the *sendpfast* function to inject packets at high rates. *sendpfast* relies on *tcpreplay* [43] for high performance packet injection.

Target device: Any WiFi-based IoT device can be used as a target. We choose Amazon Ring Spotlight Cam Battery HD Security Camera [44] for our battery drain experiments. The camera is powered by a custom 6040 mAh lithium-ion battery. The battery life of this camera is estimated to be between 6 and 12 months under normal usage [45], [46]. We leave the camera settings to their defaults which means most power-consuming options are turned off. This assures that our measurements will be an upper bound on the battery life and hence the attack might drain the battery much faster in the real world. Authors in [1] pointed out the possibility of a battery draining attack by forging beacon frames. However, they did not provide any evaluations to test this idea. Moreover, we show how sending fake packets in addition to fake beacon frames can significantly increase the power consumption on the victim device.

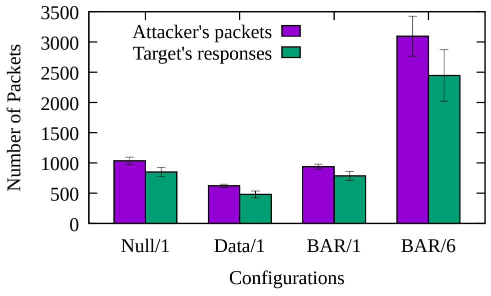
B. Results

We evaluate the effectiveness of the battery drain attack in terms of range and using different payload configuration.

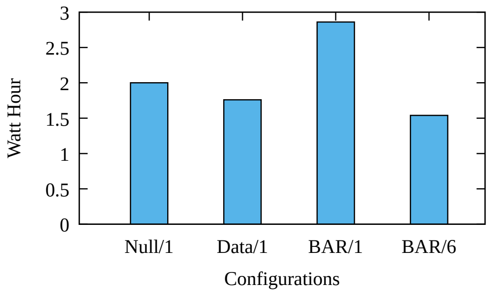
1) Finding the optimal configuration:: As discussed in VI-A1, sending block ACK requests at the lowest bitrate (i.e., 1 Mbps) should maximize the power consumption of the target device. To verify this, we have conducted a series of experiments with different types of query packets and transmission bitrates. In each experiment, we continuously transmit query packets to the Ring security camera. In all experiments, we start with a fully charged battery and the attacker injects query packets as fast as possible.

Figure 14 (a) shows the maximum number of packets the attacker could transmit to the target device, and the number of responses it receives per second. Figure 14 (b) shows

[1]For example, ESP8266 [40] and ESP32 [38] WiFi modules draw 50 and 100 mA when receiving while they draw 170 and 240 mA when transmitting. These low-power WiFi modules are very popular for IoT devices.



(a) Number of packets to/from the target device



(b) Energy consumption

Fig. 14: The Effect of packet type and transmission rate on the battery-draining attack.

the amount of energy drawn from the battery during one hour of the attack which depends on the number of response packets (i.e., Figure (a)) and the transmission time of each response packet. The transmission time is inversely related to the transmission rate. As expected, sending Block ACK Requests (BAR) drains more energy from the battery since the target device spends more time on transmission than receiving. Moreover, the results verify that although increasing the data rate from 1Mbps to 6Mbps (BAR/1 versus BAR/6) increases the number of responses, it decreases the energy drained. As mentioned before, this is because the total time spent transmitting decreases mainly due to overheads such as channel sensing and backoffs. This result confirms that sending block ACK requests (BAR) with the lowest datarate is the best option to drain the battery of the target device.

2) Battery drain with optimal configurations: We use the best setting which is a block ACK request (BAR) query transmitted at 1 Mbps to fully drain the battery of the Ring security camera. We are able to drain a fully charged battery in 36 hours. Considering the fact that the typical battery life of this camera is 6 to 12 months, our attack reduces the battery life by 120 to 240 times! It is worth mentioning that since a typical user charges the battery every 6-12 months, on average the batteries are at 40-60%, and therefore it would take much less for our attack to kill the battery. Moreover, the RING security camera is using a very large battery, most security sensors are using smaller batteries. Table IV shows the amount of time it takes to drain different batteries. For example, it takes less than 40 mins to kill a fully charged AAA battery which is a common battery in many sensors.

3) Range of WiFi battery draining attack: A key factor in the effectiveness of the battery draining attack is how far the attacker can be from the victim's device and still be able

Battery Type	Voltage	Full Capacity	100% Drain	25% Drain
CR2032 coin	3.0 V	0.68 Wh	14 m	3.5 m
AAA	1.5 V	1.87 Wh	39 m	10 m
AA	1.5 V	4.20 Wh	90 m	22 m

TABLE IV: The time to drain different types of batteries

to carry on the attack. If the attack can be done from far away, it becomes more threatening. To evaluate the range of this attack, we design an experiment in which the attacker transmits packets to the target from different distances and we measure what percentage of the attacker's packets are responded to by the target device. We use a realistic testbed. The Ring security camera is installed in front of a house, and the attacker is placed in a car, parked at different locations on the street. We test the attack at 10 different locations up to 150 meters away from the target device. Figure 15 shows these locations and our setup. Each yellow circle represents each of the locations tested at. The numbers inside the circles show the percentage of the attacker's packets responded to by the camera. Each number is an average of over 60 one-second measurements. The closest distance is about 5 meters when we park the car in front of the target house. In this location 97% of the attacker's packets are responded to. We conducted other experiments within 10 meters of the target (not shown here) and we obtained similar results. Our results show that even within a distance of 100 meters, almost all attacker's packets are responded to by the victim's device. In some locations such as the rightmost circle (at 150 meters away), we could still achieve a reply rate as high as 73%, confirming our attack works even at that distance. The reason for achieving such a long range is that the attacker transmits at a 1 Mbps bitrate which uses extremely robust modulation and coding rate (i.e. BPSK modulation and a 1/11 coding rate).

VII. Is It Possible to Stop these Attacks?

In this section, we explore potential defence mechanisms against the attacks presented in this paper. Two categories of solutions are conceivable: 1) patching the loopholes to solve the root cause of the problem 2) designing attack-specific solutions that prevent particular attacks.

A. Fixing the root cause

We now study if it is possible to patch the 802.11 loopholes namely, responding to fake packets and staying awake. Both breathing rate and battery-draining attacks rely on the fact that WiFi devices respond to the attacker's fake packets. This flaw exists in all WiFi devices even if they use the most advanced encryption mechanisms defined in the IEEE 802.11 standards.

WiFi responds when it should not: To avoid this loophole, WiFi devices must verify that a frame is legitimate before sending an acknowledgment (ACK). However, this is very challenging to do due to the timing requirements of the IEEE 802.11 standard. The standard specifies that an ACK must be sent within a Short Interframe Space (SIFS) interval, which is 10 microseconds (μs) for the 2.4 GHz band and 16 μs for the 5 GHz band. If the transmitter does not receive an ACK within the SIFS interval, it assumes that the frame was lost and retransmits it.

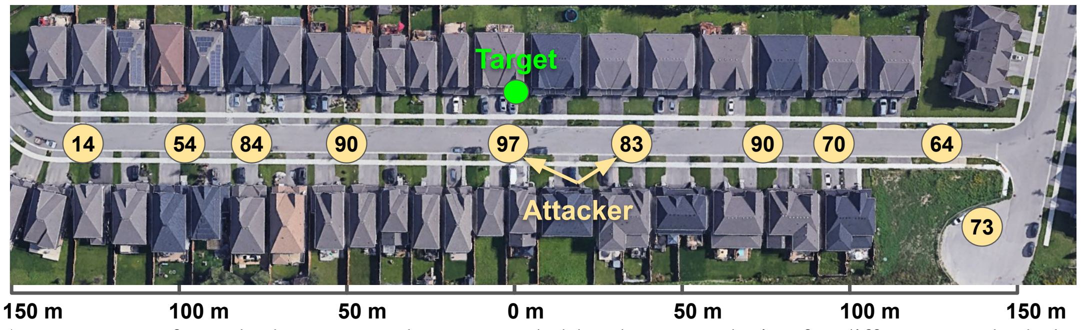


Fig. 15: Percentage of attacker's query packets responded by the target device for different attacker's locations.

To verify the legitimacy of a frame, a WiFi device must decode the frame using the encryption mechanism the network is using. This decoding process takes between 200 and 700 μs using the WPA2 security protocol [47], [48], [49], which is significantly longer than the SIFS interval. Therefore, existing WiFi devices cannot verify the legitimacy of a frame before sending an ACK. Instead, they simply acknowledge any frame that passes an error detection check.

One potential approach to patch this loophole is to implement the security decoder in WiFi hardware instead of software to significantly speed up its delay. Although this may solve the problem in future WiFi chipsets, it will not fix the problem in billions of WiFi chipsets which are already deployed. Therefore, we also propose attack-specific solutions for billions of existing WiFi devices.

WiFi stays awake when it should not: The stay awake loophole involves sending fake 802.11 beacon frames to the target device. The beacon frames do not have any encryption so that even devices that have not joined the network can understand them. The lack of encryption enables attackers to forge beacon frames and pretend to be the actual access point.

This loophole can be prevented fundamentally by adding an integrity checking mechanism to beacons. The WPA3 security standard has introduced beacon protection which allows WiFi devices to detect fake beacons. However this optional feature has not been implemented on any chipset yet and it may never become a popular feature because of lack of backward compatibility with billions of existing WiFi devices[50]. As a result, other solutions are needed for attacks that are made possible because of this loophole.

B. Attack-specific defence mechanisms

Breathing rate detection attack: In the previous sections, we have shown how effective breathing rate detection attack is. A natural question is whether it is possible to stop an attacker from monitoring a person's breathing rate.

Even if someone finds out an attack is happening by monitoring and checking for fake packets, the physical layer still sends back ACKs for received packets. Therefore, the attack cannot be stopped at the root. Despite this limitation, we present a solution that confuses the attacker and can potentially prevent it from estimating the breathing rate accurately.

As explained before, breathing rate attack relies on the CSI changes of WiFi signals to estimate the breathing rate. Therefore, one possible solution to stop such an attack is to artificially create similar changes in the CSI. For example, if the target WiFi device periodically changes its transmission power, it might impact the CSI, hence preventing the attacker from estimating the breathing rate.

Although this approach is plausible, changing the transmission power frequently can significantly impact the throughput of WiFi devices. This is because changing the transmission power impacts the SNR of the link, as a result, it can cause packet drops. Moreover, it may confuse the rate adaptation algorithm on the target device to believe that the channel changes very dramatically over time which can further impact the throughput. However, it might be possible to change the transmission power by a small amount to stop the attack, while the throughput is not impacted significantly. Next, we try to find the minimum required change in the transmission power to disrupt the breathing rate attack.

We run a set of experiments where the target device is changing its transmission power. We examine different transmission power changes at different intervals. In all experiments, there is a person next to the target device and the attacker tries to estimate the breathing rate. Figure 16 shows the results of this experiment. In particular, the figure shows whether the attacker was able to successfully detect the target person's breathing rate for a given amount of change in the transmission power and a given period of change. The shaded area in the plot shows the configurations under which the defense works and the attacker is unable to detect breathing.

These results show that if we lower the interval (i.e. increase the frequency of change), the required change in the transmission power to prevent the attack decreases. However, note that lowering the intervals below 0.5 s (i.e. 60 times per minute) will make the defense ineffective. This is due to the fact that an adult's breathing rate is in the range of 12-20 breaths per minute and a baby's breathing rate is in the range of 40-60 breaths per minute [51]. Therefore, the attacker can easily filter out any changes which are above 60 times per minute. Hence, the optimal way to prevent the attack is to make the target device's transmission power change by 3 dB every 0.5 s.

Now, the next question is whether such a change in the transmission power impacts the throughput of WiFi devices. To evaluate this hypothesis, we set up two laptops. One is a server acting as a WiFi Access Point (AP), and the other one is a client acting as a target device. We use iperf [52] to send a stream of UDP packets from the client to the server and to monitor the throughput for 10 minutes. We set UDP data rate on the target device to 20 Mbps and perform this

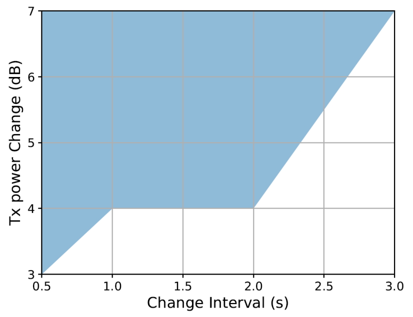


Fig. 16: The effectiveness of the proposed defence for different interval and transmit power changes. Shaded area shows when the defense is effective.

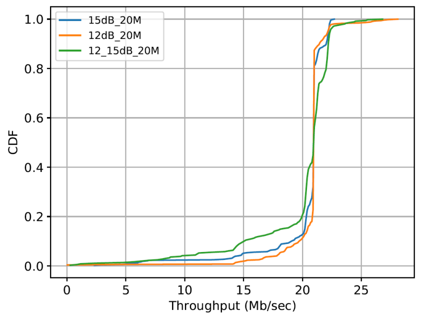


Fig. 17: The impact of the proposed defence technique on network throughput when the application data rate is set to 20 Mbps.

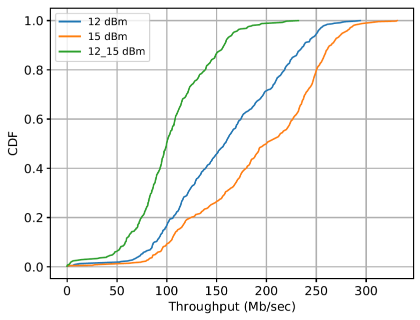


Fig. 18: The impact of the proposed defence technique on network throughput when there is no limit set for the application data rate (saturating the channel).

experiment in three different scenarios: (1) the Tx power is stable at 12 dB, (2) the Tx power is stable at 15 dB, and (3) the Tx power oscillates between 12 dB and 15 dB every 0.5 s.

Figure 17 shows the CDF of throughput for all three scenarios. The figure shows that the WiFi device achieves roughly the same throughput in all scenarios. In particular, oscillating the transmission power does not impact the throughput of the device. Although the 20 Mbps data rate used in this experiment is more than what most WiFi devices use, there are applications that require higher bandwidths. Therefore, next, we evaluate if oscillating the transmission power impact the throughput of WiFi devices when the device transmits at the maximum data rate (i.e., saturating the WiFi channel).

Figure 18 shows the CDF of throughput when the client WiFi device saturates the channel for all three scenarios. As expected, lowering the transmission power from 15 dBm to 12 dBm reduces the throughput. However, the results show that when the transmission power oscillates between 12 and 15 dBm, the throughput is even lower. The reason for this is that WiFi uses a rate adaptation algorithm that changes the physical-layer bitrate according to the channel conditions. When we change the transmission power back and forth, the algorithm tries to adapt constantly. However, these repeated artificial changes cause the rate adaptation algorithm to aggressively reduce the bitrate or to choose a bitrate that is too high which results in excessive packet drop. As a result, the throughput suffers significantly.

In summary, although our idea of changing the transmission power prevents an attacker from monitoring breathing rate, it potentially impacts the throughput especially when high application data rates are used. Therefore, WiFi devices can change the transmission power when they do not need high bandwidth. However, as we approach the limit of the channel, this problem becomes a trade-off between privacy and throughput.

Battery-draining attack A WiFi device can detect this attack if it receives too many beacon frames forcing it to stay awake. A WiFi access point typically sends out beacon frames every 102.4 ms. Therefore, if a device suddenly starts receiving more beacons, it is a strong indicator of this type of attack. Although this attack is possibly easy to detect, it might be challenging to react to it properly. This is because the power saving mechanism is typically implemented in the hardware and there might not be an easy way to patch existing WiFi devices.

A potential solution for this loophole is that WiFi chipsets can check to see if the access point sends any encrypted data packet after claiming to have buffered packets for the client. If no packets is received after multiple false claims, the device should go to sleep to prevent staying away for long periods of time under attack.

VIII. Ethical Considerations

We discussed our project and experiments with our institutions' IRB office and they determined that no IRB review nor IRB approval is required. Moreover, the house and WiFi devices used in most experiments are owned and controlled by the authors. Finally, in order to expedite mitigating the attacks presented in this paper, we have started engagements with WiFi access point and chipset manufacturers.

IX. Conclusion

In this work, we identify two loopholes in the WiFi protocol and demonstrate their possible privacy and security threats. In particular, we reveal that today's WiFi radio responds to packets from unauthorized devices outside of the network and it can be easily manipulated to keep awake. These loopholes can be exploited by malicious attackers to jeopardize our daily use of WiFi devices. As examples, we demonstrate how an attacker can take advantage of these loopholes to extract private information such as breathing rate and quickly exhaust the battery of a typical IoT device, leaving the victim's device in a disabled state.

References

[1] M. Vanhoef, P. Adhikari, and C. Pöpper, "Protecting wi-fi beacons from outsider forgeries," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 155–160.

[2] M. Agarwal, D. Pasumarthi, S. Biswas, and S. Nandi, "Machine learning approach for detection of flooding DoS attacks in 802.11 networks and attacker localization," *International Journal of Machine Learning and Cybernetics*, vol. 7, pp. 1035–1051, 2016.

[3] B. Alotaibi and K. Elleithy, "Rogue Access Point Detection: Taxonomy, Challenges, and Future Directions," *Wireless Personal Communications*, vol. 90, pp. 5021– 5028, 10 2016.

[4] J. Bellardo and S. Savage, "802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," *Proceedings of 12 USENIX Security Symposium*, 2003.

[5] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman, "Beacon-stuffing: Wi-fi without associations," in *Eighth IEEE Workshop on Mobile Computing Systems and Applications*, 2007, pp. 53–57.

[6] S. Zehl, N. Karowski, A. Zubow, and A. Wolisz, "Lows: A complete open source solution for wi-fi beacon stuffing based location-based services," in *2016 9th IFIP Wireless and Mobile Networking Conference (WMNC)*, 2016, pp. 25–32.

[7] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions." in *USENIX security symposium*, vol. 12. Washington DC, 2003, pp. 2–2.

[8] M. Vanhoef, P. Adhikari, and C. Pöpper, "Protecting wi-fi beacons from outsider forgeries," ser. WiSec, 2020.

[9] "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 4: Protected Management Frames," *IEEE Std 802.11w-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, and IEEE Std 802.11y-2008)*, pp. 1–111, 2009.

[10] G. Lan, M. F. Imani, Z. Liu, J. Manjarrés, W. Hu, A. S. Lan, D. R. Smith, and M. Gorlatova, "MetaSense: Boosting RF Sensing Accuracy Using Dynamic Metasurface Antenna," *IEEE Internet of Things Journal*, vol. 8, 2021.

[11] Y. Ma, G. Zhou, and S. Wang, "Wifi sensing with channel state information: A survey," *ACM Computing Surveys*, vol. 52, no. 3, 2019.

[12] F. Adib, H. Mao, Z. Kabelac, D. Katabi, and R. C. Miller, "Smart homes that monitor breathing and heart rate," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 2015, pp. 837–846.

[13] H. Wang, D. Zhang, J. Ma, Y. Wang, Y. Wang, D. Wu, T. Gu, and B. Xie, "Human respiration detection with commodity wifi devices: Do user location and body orientation matter?" ser. UbiComp, 2016, p. 25–36.

[14] J. Liu, Y. Wang, Y. Chen, J. Yang, X. Chen, and J. Cheng, "Tracking vital signs during sleep leveraging off-the-shelf wifi," ser. MobiHoc, 2015.

[15] Y. Zheng, Y. Zhang, K. Qian, G. Zhang, Y. Liu, C. Wu, and Z. Yang, "Zero-effort cross-domain gesture recognition with wi-fi," ser. MobiSys, 2019, p. 313–325.

[16] R. H. Venkatnarayan, G. Page, and M. Shahzad, "Multi-user gesture recognition using wifi," ser. MobiSys, 2018.

[17] A. Virmani and M. Shahzad, "Position and orientation agnostic gesture recognition using wifi," ser. MobiSys, 2017.

[18] Q. Pu, S. Gupta, S. Gollakota, and S. Patel, "Whole-home gesture recognition using wireless signals," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, 2013, pp. 27–38.

[19] Y. Zhu, Z. Xiao, Y. Chen, Z. Li, M. Liu, B. Y. Zhao, and H. Zheng, "Et tu alexa? when commodity wifi devices turn into adversarial motion sensors," in *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[20] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for ad-hoc wireless networks," in *International workshop on security protocols*. Springer, 1999, pp. 172–182.

[21] L. Caviglione and A. Merlo, "The energy impact of security mechanisms in modern mobile devices," *Network Security*, vol. 2012, no. 2, pp. 11–14, 2012.

[22] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 105–114.

[23] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff, "Hypnos: understanding and treating sleep conflicts in smartphones," in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013, pp. 253–266.

[24] U. Fiore, F. Palmieri, A. Castiglione, V. Loia, and A. De Santis, "Multimedia-based battery drain attacks for android devices," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. IEEE, 2014, pp. 145–150.

[25] U. Fiore, A. Castiglione, A. De Santis, and F. Palmieri, "Exploiting battery-drain vulnerabilities in mobile smart devices," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 90–99, 2017.

[26] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. IEEE, 2008, pp. 296–296.

[27] A. Abedi and O. Abari, "Wifi says "hi!" back to strangers!" in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets*, 2020, p. 132–138.

[28] A. Abedi and D. Vasisht, "Non-cooperative wi-fi localization and its privacy implications," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (MobiCom*, 2022, p. 570–582.

[29] Philippe Biondi, *Scapy*, 2020, https://scapy.net/.

[30] G. Combs, *Wireshark*, 2020, https://www.wireshark.org/.

[31] *Wi-Fi Driver*, Espressif Systems, 6 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html

[32] *IDF Monitor*, Espressif Systems, 6 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/tools/idf-monitor.html

[33] "Polite wifi experiment traces," https://github.com/LuHaofan/Polite-WiFi-Datasets.

[34] S. Arshad, C. Feng, Y. Liu, Y. Hu, R. Yu, S. Zhou, and H. Li, "Wi-chase: A wifi based human activity recognition system for sensorless environments," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2017, pp. 1–6.

[35] Y. Gu, J. Zhan, Y. Ji, J. Li, F. Ren, and S. Gao, "Mosense: An rf-based motion detection system via off-the-shelf wifi devices," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2326–2341, 2017.

[36] H. Abdelnasser, K. Harras, and M. Youssef, "A ubiquitous wifi-based fine-grained gesture recognition system," *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2474–2487, 2019.

[37] E. Dafna, A. Tarasiuk, and Y. Zigel, "Sleep-wake evaluation from whole-night non-contact audio recordings of breathing sounds," *PloS one*, vol. 10, no. 2, p. e0117382, 2015.

[38] *ESP32 datasheet*, Espressif Systems, 4 2019, https://www.espressif.com/sites/default/files/documentation/\esp32_datasheet_en.pdf.

[39] A. Abedi, F. Dehbashi, M. H. Mazaheri, O. Abari, and T. Brecht, "Witag: Seamless wifi backscatter communication," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2020, pp. 240–252.

[40] *ESP8266 datasheet*, Espressif Systems, 4 2020, https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[41] "Rtl8812au," https://www.realtek.com/en/products/communications-network-ics/item/rtl8812au.

[42] "aircrack-ng/rtl8812au," https://github.com/aircrack-ng/rtl8812au.

[43] "Tcpreplay - pcap editing and replaying utilities," https://tcpreplay.appneta.com/.

[44] "Ring spotlight cam battery," https://ring.com/products/spotlight-cam-battery", Amazon.

[45] "Ring spotlight cam battery review," https://www.security.org/security-cameras/ring/review/spotlight-cam-battery/, security.org.

[46] "Ring spotlight cam battery review," https://www.pcmag.com/reviews/ring-spotlight-cam-battery, pcmag.

[47] S. S. Kolahi and A. A. Almatrook, "Impact of security on bandwidth and latency in IEEE 802.11ac client-to-server WLAN," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 893–897.

[48] P. Li, S. S. Kolahi, M. Safdari, and M. Argawe, "Effect of WPA2 Security on IEEE 802.11n Bandwidth and Round Trip Time in Peer-Peer Wireless Local Area Networks," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, 2011, pp. 777–782.

[49] M. Saleh, J. Gaber, and M. Wack, "Sensor Networks Applications Performance Measures for IEEE802.11n WiFi Security Protocols," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ser. ICFNDS '17, 2017.

[50] D. Schepers, A. Ranganathan, and M. Vanhoef, "On the Robustness of Wi-Fi Deauthentication Countermeasures," ser. WiSec '22, 2022, p. 245–256.

[51] S. C. Health. Breathing problems. https://www.stanfordchildrens.org/en/topic/default?id=breathing-problems-90-P02666.

[52] J. Dugan, "Iperf: The ultimate speed test tool for tcp, udp and sctp," http://sourceforge.net/projects/iperf/, 2020.