Sniffing Wireless Keyboard Traffic

Deep Majumder
19CS3015

November 2023

1 Introduction

Wireless keyboards are commonly used as an alternative to standard wired keyboards, owing to the advantage they provide in cable-management as well as for portability. Wireless keyboards typically use one of the following two (or both are available as options) protocols for communicating with the computer:

1. **Enhanced ShockBurst (ESB)**: This is a proprietary 2.4 GHz RF protocol, and are used in keyboards that have a separate USB receiver. Both the keyboard and the receiver have a symmetric transmitter/receiver SoC, with the most common models being Nordic nRF24L01+, TI CC500 or Cypress CYRF6936.

2. **Bluetooth Low Energy (BLE)**: This is present on more expensive keyboards, owing to the fact that BLE modules are significantly more expensive than ESB modules (₹350 vs ₹65). These keyboards do not require separate receivers.

The BLE protocol is sufficiently guarded, at the protocol level, against simple sniffing attacks. On the other hand, the ESB protocol is entirely vulnerable to sniffing attacks, since the protocol doesn't require any form of encryption of payload data. Sniffing 2.4 GHz traffic can be easily done with software-defined radio receivers, with the only downside being that such equipment is fairly expensive (HackRF One, a commonly used software-defined radio by hardware hackers, costs around ₹23k). However, Samy Kamkar has found [2] a method to exploit a bug in the implementation of the nRF24L01+ SoC, which allows us to sniff ESB packets using just this SoC and a microcontroller board. This method will form the base of our investigation.

2 Goals

Since the basic technique is available in [2] and the code is already available in [3], we have aimed to:

- Optimize the code, in particular to improve the time required to discover keystrokes.
- Estimate the time required to start discovering keystrokes as a function of typing speed.

3 Threat Model

The threat model that we are considering for this attack is as follows:

- **Attacker action**: The attacker will be acting **passively** by sniffing keyboard traffic.
- **Attacker capability**: The attacker need not have much financial resources, since the total hardware cost is less than ₹1000 (₹300 for a clone Arduino board, ₹65 for nRF24L01+, ₹250 for a SPI flash chip, the rest for casing and wire). The code is fairly simple and available.
- **Attacker access**: The attack hardware needs to be in physical proximity of the victim, since RF signals fall off rapidly with inverse-square relation. On the other hand, since it is a passive attack, the attacker need not be constantly present. Ideally, the attacker needs two accesses - once to drop off the attack hardware and once again to retrieve it. In addition, with somewhat more sophisticated microcontrollers, it is possible to transmit the information online (a microcontroller with a WiFi module), in which case the attacker only needs one access.

Under this attack model, an attacker can record any information that is typed by the user, including passwords.

4 Enhanced ShockBurst Protocol

The packet format for Enhanced ShockBurst protocol, as specified in [4], is shown in Figure 1. A packet has the following components:

1. **Preamble**: This must be either 0xAA or 0x55, essentially forming a one byte sequence of alternating bits, which is used to demarcate the beginning of the packet.
2. **Address**: The MAC address must be “legally” between 3 and 5 bytes. The transmitter will transmit packets with this field set. The receiver must be provided with the expected MAC address, and it uses the preamble combined with the MAC address to identify valid packets. All packets not matching this are dropped. In particular, [4] specifies that the address must not start with 0xAA or 055.

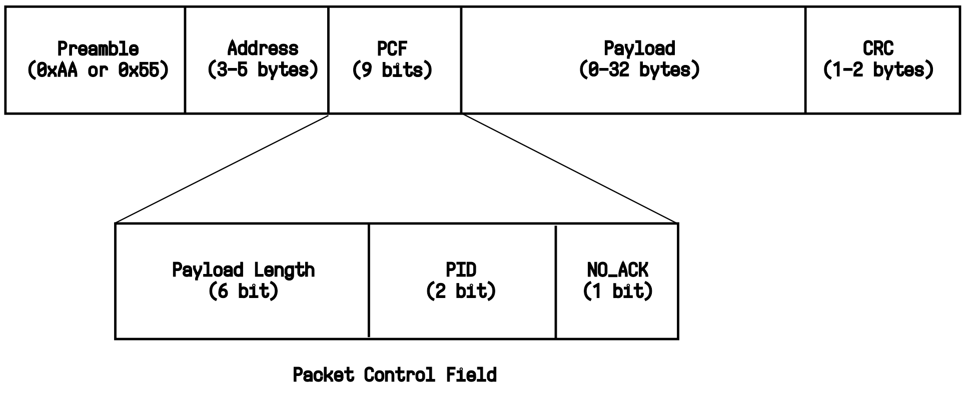


Figure 1: Packet format for Enhanced ShockBurst protocol

3. **PCF**: The packet control field contains the payload length, a packet identifier (PID) field as well as the `NO_ACK` bit. The `NO_ACK` bit tells the receiver that the packet must not be auto-acknowledged. The PID field is auto-incremented from the transmitter to help the receiver identify new packets from re-transmitted packets (which happens when auto-acknowledgement is enabled and ACK packet is lost).

4. **Payload**: The payload as can be seen is entirely user defined, with no provision for encryption in the protocol. Some keyboards use hardcoded keys to encrypt the payload with AES, others use some simple insecure form of "encryption" while the cheapest keyboards simply send the packets unencrypted.

5. **CRC**: The CRC polynomial for 1-bit CRC is X^8+X^2+X+1 with initial value 0xFF, while for 2-bit CRC is $X^{16}+X^{12}+X^5+1$ with initial value 0xFFFF. Packets are dropped if CRC check fails.

5 Sniffing using nRF24L01+

The major obstacle from using the nRF24L01+ for sniffing packets is that it *requires the MAC address to be known* to receive packets at all. If we know the MAC address somehow, then we can passively sniff all packets by disabling acknowledgement from the receiver side and disabling CRC checking. Samy Kamkar mentions a workaround this in [2], which was originally published by Travis Goodspeed in [1].

Figure 2 shows the main idea presented in [1]. The shaded rectangles shows how the stream of bytes shown is transmitted and is expected to be interpreted by the receiver. On the other hand, the dotted rectangles show how the stream of bytes might be interpreted if we get a bit "lucky". In that case, we do not need to know the MAC address at all, which is instead captured as a part of the payload.

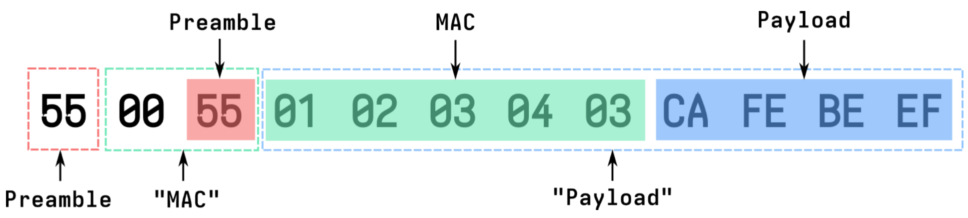


Figure 2: Dual interpretation of ESB packet, which allows sniffing (adapted from [1])

The luck required is that there must be an additional 0x55 present just before this byte stream. In practice, this isn't all that uncommon since such alternating-bit preambles are ubiquitous in most RF devices and so are often floating about (which is why the MAC address is also used as an extended preamble). In addition, and perhaps more importantly, it requires the *MAC address' length to be set to 2*, which is on-paper illegal.

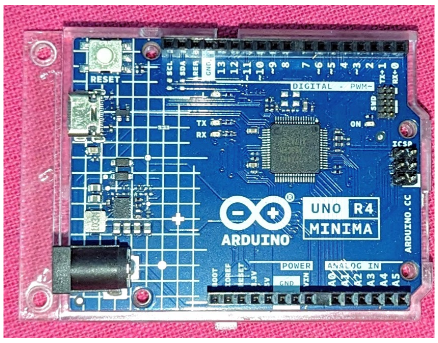
According to [4], the MAC address length is set by the 2-bit `AW` register, which can have the values as specified in Table 1. Although setting `AW` to `00` is specified as *illegal*, it turns out that the chip implementation is buggy and on setting it, the **address length becomes 2**. This means, we can use the idea shown in Figure 2 to "probabilistically" sniff packets.

Table 1: `AW` register interpretation

Bits	Interpretation
`00`	*Illegal*
`01`	3-byte address
`10`	4-byte address
`11`	5-byte address

6 Experimental Setup

For performing the experiments, we have used a nRF24L01+ transceiver, created using a nRF24L01+ board and an Arduino microcontroller board, as shown in Figure 3. The nRF24L01+ board has the nRF24L01+ SoC, an SMD antenna as well as male 2.54 mm header pins to provide connection over SPI. Figure 3a shows the Arduino Minima board used to create one transceiver while Figure 4 shows the Arduino WiFi board used to create the other one. These boards are essentially the same for the purposes of our experiments. Figure 4 also shows the full transceiver, with the nRF24L01+ board connected to the Arduino board with DuPont female-to-male cables.


(a) Arduino Minima

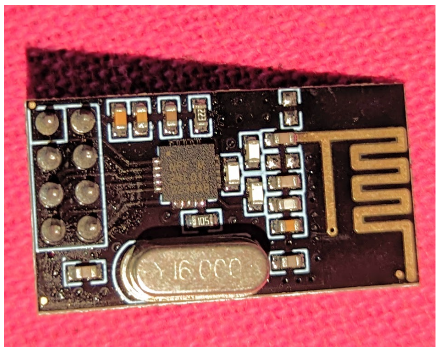

(b) nRF24L01+ board

Figure 3: Components used to create the 2.4GHz Enhanced ShockBurst transceiver

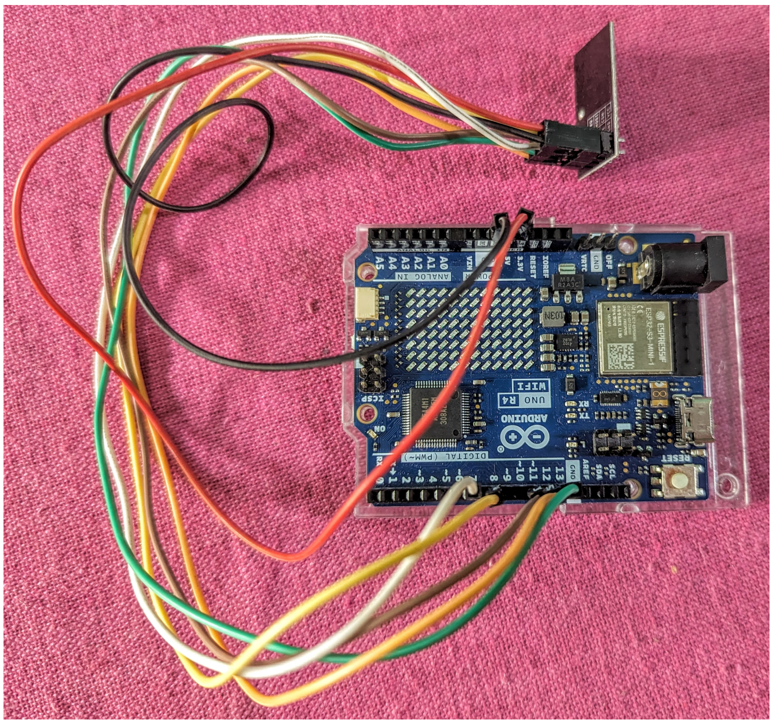


Figure 4: The 2.4 GHz Enhanced ShockBurst transceiver, with the nRF24L01+ board connected to a Arduino WiFi board over SPI

6.1 Keyboard Emulation

Since we aim to measure the time required to begin sniffing as a function of typing speed, it is important for us to be able to control typing speed and keep it consistent throughout the period of measurement. This is fairly difficult for a person to do with a physical keyboard. Therefore, we decided to emulate keystrokes by making a second transceiver, which will send keystrokes at a given typing speed. A second Arduino board was required for this since the Arduino Uno Rev4 boards have only a single SPI port and so can connect to only one nRF24L01+ board at a time.

While it is possible to sniff the packets from any wireless keyboard using ESB protocol by the technique explained in Section 5, to actually read keystrokes from the payload requires us to reverse engineer the payload format used by the keyboard vendor. [5] has reverse engineered the payload format of Microsoft and Logitech keyboards. While the payload format is ultimately important to recover the keystrokes, it is not relevant for the measurements we are doing. Therefore, we have assumed the Microsoft packet format to send keystrokes.

To generate keystrokes, we first fix the typing speed in words per minute (WPM). Assuming an average word length of 5, following the estimate made in [6] of 5.1 characters, we calculate the expected characters per minute (CPM). This allows us to calculate the expected delay between characters and we have assumed a 20 ms variance in that delay. The actual delay between characters is sampled from a Gaussian distribution with $\mu = \text{CPM}$ and $\sigma = 20$. In addition, people tend to type in bursts at any WPM, rather than continuously. To account for that, we uniformly sample a burst length in the range $[30, 50]$ and at the end of each burst, we add an additional delay of 50 ms. The character being sent is uniformly sampled from the space comprising all lowercase character, all digits and the space character.

6.2 Sniffing Protocol

To use the technique mentioned in Section 5, we need to additionally know the channel in which the keyboard is sending the keystrokes. This channel for keyboards that are sold in the US must range between 3 and 80, as required by the FCC. Since keyboards can be assumed to share the same firmware globally, Indian keyboards can also be assumed to follow the same range. In order to detect the channel to sniff on eventually, we need to cycle through all the channels until we get a hit and then continue sniff on those channels. In order to randomize the order of channels been cycled through, we permute the order of channels at the end of each cycle. Additionally, we need to decide the amount of time spent on each channel. Empirically, we have found that if we sample the wait time (in millisecond) from a Gaussian distribution with $\mu = 500$ and $\sigma = 50$, it is effective for the typing speeds we are exploring.

Table 2: Hit Time and Hit Count vs WPM statistics

WPM	**Time (s)**			**Hits**	
	Mean	**Std. Dev.**	**Median**	**Count**	**Spurious (%)**
20	194	106	166	19	31.6
30	87	70	73	16	25.0
40	74	76	53	24	12.5
50	53	39	53	34	8.8
60	51	54	40	36	11.1
70	31	35	23	55	7.3

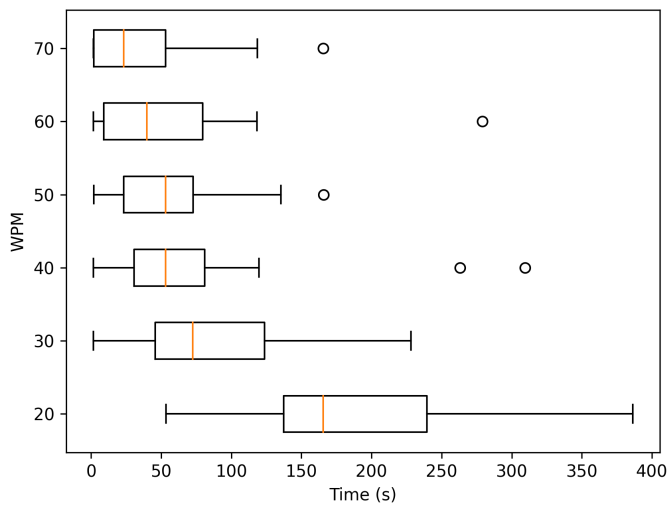


Figure 5: Box-and-whisker plot showing the time to crack vs the WPM

7 Results

We measured the time required for the first packet to be sniffed by the receiver at various WPM's. The resulting data is summarised in Table 2. It is to be noted that the large standard deviation, which is almost of the order of the mean, indicates that a large part of the distribution lies in the tails. Therefore, the median is a better statistic for the expected time to sniff. As can be seen in Figure 5, the median time for sniffing reduces with increasing time, except for the case of 50 WPM (which is probably due to the presence of outliers). In Table 2, we also record the number of hits per WPM and the percent of spurious hits. Spurious hits occur because of other RF packets that may be present in the area at the same time. As expected, with the increase in frequency of valid packets (with increasing WPM), the spurious hit count again decreases.

Table 3: Hits vs received channel (where sender channel is 34)

Channel	Count
34	17
35	76
38	51
39	16

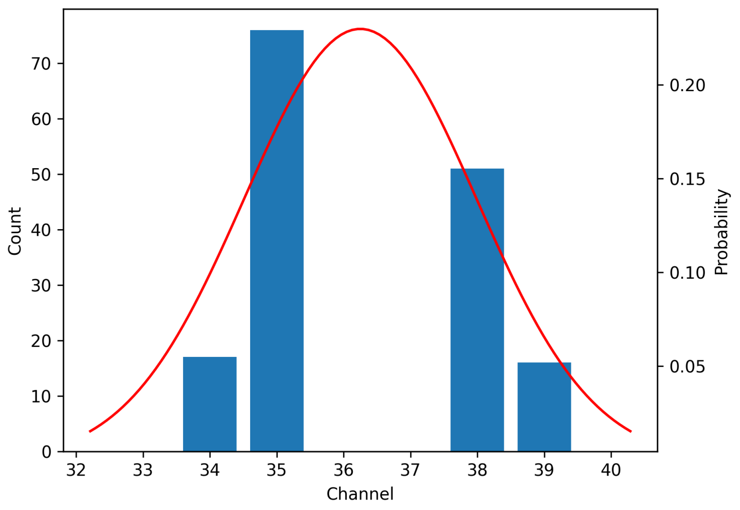


Figure 6: Hit count vs channel plot superimposed with a plot of Gaussian distribution with mean and standard deviation of the count data

In Table 3, we record the channel on which we have received the packets, while sending packets on channel 34. Surprisingly, most of the packets are received on channel 35. If we plot the count vs channel bar diagram and superimpose it with a plot of the Gaussian distribution with mean and standard deviation obtained from the channel hit distribution (Figure 6), we can see that the peak of the distribution is around 36.25. The rest of the bars also roughly correspond to the distribution. It is almost as though the sender is sending at channel 36 and the reception is only on sidebands. This may be correlated with the fact that we are setting a register to an illegal value.

8 Code

The Arduino code of the sender is hosted at nrf24_transmit, while that of the receiver is hosted at nrf24_sniff. The Jupyter notebook of the analysis is hosted at design-lab-analysis.

References

[1] Travis Goodspeed. *Promiscuity is the nRF24L01+'s Duty.* Feb. 7, 2011. URL: http://travisgoodspeed.blogspot.com/2011/02/promiscuity-is-nrf24l01s-duty.html (visited on 11/20/2023).

[2] Samy Kamkar. *Keysweeper.* Jan. 12, 2015. URL: https://samy.pl/keysweeper (visited on 11/20/2023).

[3] Samy Kamkar. *keysweeper.* Github. Jan. 12, 2015. URL: https://github.com/samyk/keysweeper (visited on 11/20/2023).

[4] *nRF24L01+ Single Chip 2.4 GHz Transceiver. Preliminary Product Specification v1.0.* Nordic Semiconductor. Mar. 2008.

[5] Thorsten Schroeder and Max Moser. *Practical Exploitation of Modern Wireless Devices.* DreamLab Technologies. Mar. 24, 2010. URL: http://www.remote-exploit.org/content/keykeriki_v2_cansec_v1.1.pdf (visited on 11/21/2023).

[6] Bengt Sigurd, Mats Eeg-Olofsson, and Joost Van Weijer. "Word length, sentence length and frequency–Zipf revisited". In: *Studia linguistica* 58.1 (2004), pp. 37–52.