

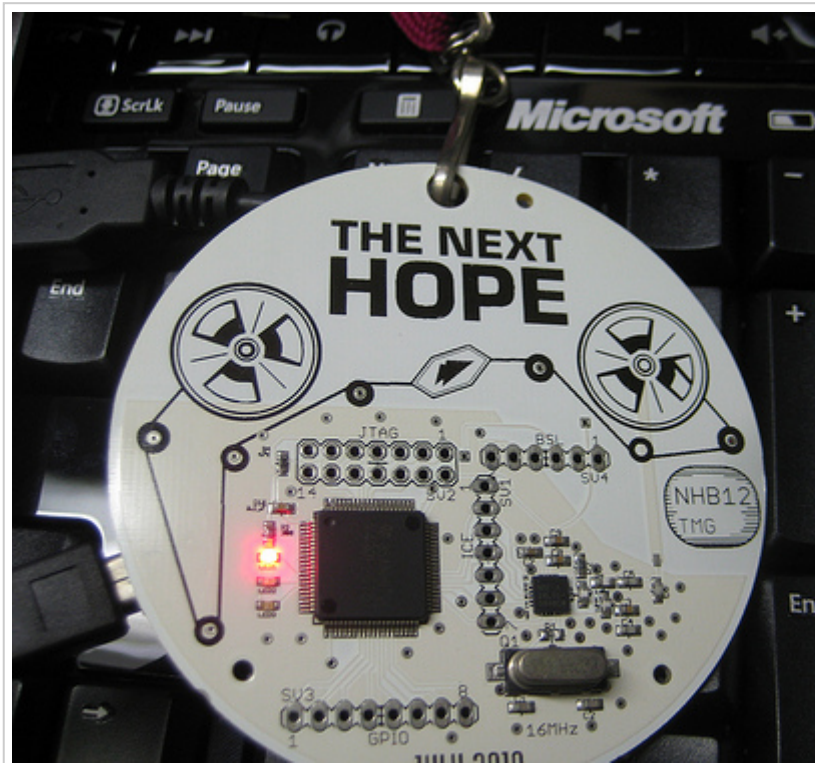
[More](#)[Next Blog»](#)[Create Blog](#)[Sign In](#)

# TRAVIS GOODSPEED'S BLOG

MONDAY, FEBRUARY 7, 2011

## Promiscuity is the nRF24L01+'s Duty

by Travis Goodspeed <travis at radiantmachines.com>  
 extending the work of Thorsten Schröder and Max Moser  
 of the KeyKeriki v2.0 project.



Similar to Bluetooth, the protocols of the Nordic VLSI nRF24L01+ chip are designed such that the MAC address of a network participant doubles as a SYNC field, making promiscuous sniffing difficult both by configuration and by hardware. In this short article, I present a nifty technique for promiscuously sniffing such radios by (1) limiting the MAC address to 2 bytes, (2) disabling checksums, (3) setting the MAC to be the same as the preamble, and (4) sorting received noise for valid MAC addresses which may later be sniffed explicitly. This method results in a rather high false-positive rate for packet reception as well as a terribly high drop rate, but once a few packets of the same address have been captured, that address can be sniffed directly with normal error rates.

As proof of concept, I present a promiscuous sniffer for the Microsoft

BLOG ARCHIVE

▶ 2013 (1)

▶ 2012 (3)

▼ 2011 (7)

▶ Dec (1)

▶ Sep (2)

▶ May (1)

▶ Mar (1)

▼ Feb (1)

Promiscuity is the nRF24L01+'s  
Duty

▶ Jan (1)

▶ 2010 (12)

▶ 2009 (29)

▶ 2008 (39)

▶ 2007 (5)

ABOUT ME

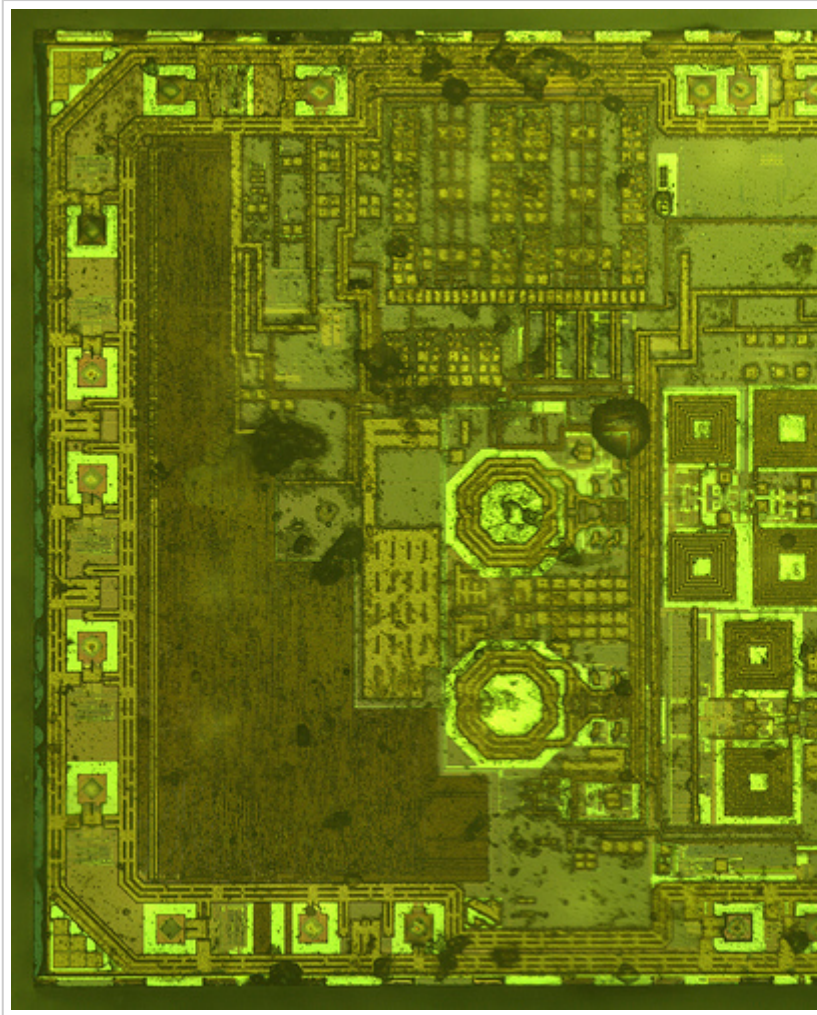


TRAVIS GOODSPEED

[VIEW MY COMPLETE  
PROFILE](#)

Comfort Desktop 5000 and similar 2.4GHz wireless keyboards. This vulnerability was previously documented at CanSecWest by Thorsten Schröder and Max Moser, and an exploit has been available since then as part of the KeyKeriki v2.0 project. My implementation differs in that it runs with a single radio and a low-end microcontroller, rather than requiring two radios and a high-end microcontroller. My target hardware is the conference badge that I designed for the Next Hope, running the GoodFET Firmware.

### Part 1: or, Why sniffing is hard.



Radio packets usually begin with a preamble, followed by a SYNC field. In the SimpliCI protocol, the SYNC is 0xD391, so the radio packets will begin with {0xAA,0xD3,0xD91} or {0x55,0xD3,0x91}. The AA or 55 in the beginning is a preamble, which is almost universally a byte of alternating 1's and 0's to note that a packet is beginning, followed by the SYNC field which makes sure that the remainder of the packet is byte-aligned.

In the case of the Nordic radios, there is no SYNC pattern unique to the radio. Instead, the MAC address itself serves this purpose. So an OpenBeacon packet will begin with {0x55, 0x01, 0x02, 0x03, 0x02,

0x01} while a Turning Point Clicker's packets will begin with {0x55, 0x12, 0x34, 0x56}. The preamble in this case will be 0x55 if the first bit of the SYNC/MAC is a 0 and 0xAA if the first bit is a 1. To make matters worse, the chip does not allow a MAC shorter than three bytes, so previously it was believed that at least so many bytes of the destination address must be known in order to receive a packet with this chip.

Moser and Schröder solved this problem by using an AMICCOM A7125 chip, which is a low-level 2FSK transceiver, to dump raw bits out to an ARM microcontroller. The ARM has just enough time to sample the radio at 2Mbps, looking for a preamble pattern. If it finds the pattern, it fills the rest of register memory with the remaining bits and then dumps them to the host by USB. In this manner, every prospective MAC address can be found. Once the address is known, KeyKeriki places that address in its second radio, an nRF24L01+, which is used to sniff and inject packets.

A similar solution is used in Michael Ossmann's Project Ubetooth sniffer for Bluetooth. See that project's documentation and Ossmann's Shmoocon 2011 video for a more eloquent explanation of why sniffing without a known SYNC is so hard.

### Part 2: or, Sniffing on the cheap.

My trick for sniffing promiscuously involves a few illegal register settings and the expectations of background noise. You can find code for this in the AutoTuner() class of the goodfet.nrf client.

First, the length of the address to sniff must be reduced to its absolute minimum. The datasheet claims that the lowest two bits of register 0x03 are responsible for address width, and that the only valid lengths at 3 bytes (01b), 4 bytes (10b), or 5 bytes (11b). Setting this value to 00b gives a 2 byte match, but when checksums are disabled, this results in a deluge of false-positive packets that appear out of background noise.

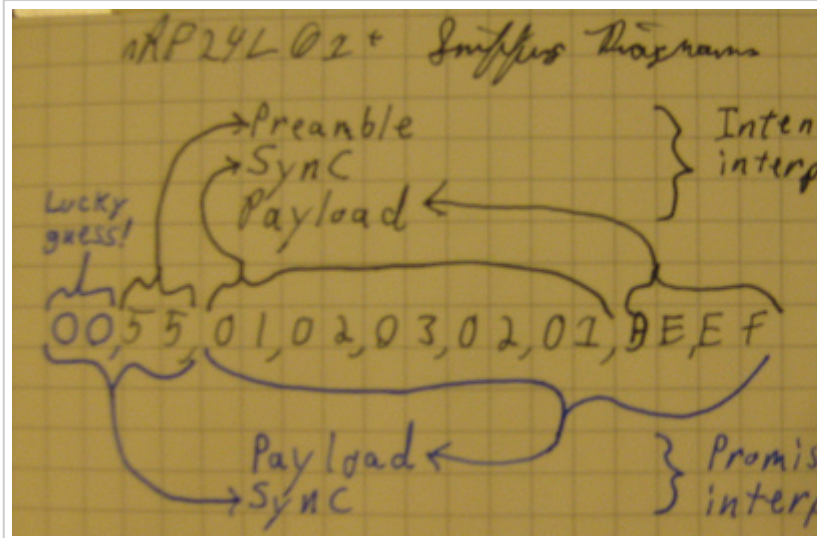
AW	1:0	11	R/W	RX/TX Address field width
				'00' - Illegal
				'01' - 3 bytes
				'10' - 4 bytes
				'11' - 5 bytes

Second, it is necessary to begin receiving before the SYNC field appears, as the Nordic chips drop the address of incoming packets, leaving only the payload. By looking at the noise returned when the address is at its shortest length, it is clear that background noise includes a lot of 0x00 and 0xFF packets as well as 0xAA and 0x55 packets, which are likely feedback from an internal clock. What then would happen if the address were to be 0x00AA or 0x0055?

What happens is that noise activates the radio a bit early, which then

syncs to the real preamble, leaving the SYNC field as the beginning of the packet payload! This is because the preamble and SYNC do not need to be immediately adjacent; rather, the SYNC can be delayed for a few bytes from the preamble in order to allow for longer preambles in noisy environments.

As a concrete example, an OpenBeacon packet looks something like the following. The SYNC field is 0x0102030201, so the packet will be cropped from that point backward. 0xBEEF is all that will be returned to the application, with everything prior to that cropped.



By making the address be 0x0055 and disabling checksums, that same packet will sometimes be interpreted as shown on the bottom. The preamble will be mistaken for a SYNC, causing the real SYNC value to be returned as the beginning of the payload. In that way, I am able to determine the SYNC/MAC field without any prior knowledge or brute force exploration.

This does depend upon the preamble being preceded by 0x00, which occurs often in background noise but is not broadcast by the attacker. So the odds of receiving a packet, while significantly worse than we'd like, are much better than the  $1/2^{16}$  you might assume. In experiments, one in twenty or so real packets arrive while a significant number of false positives also sneak in.

Recalling that the MAC addresses are three to five bytes long, and that radio noise is rather distinct, it stands to reason that noise can easily be separated from real packets by either manually checksumming to determine packet correctness or simply counting the occurrences of each address and taking the most popular. You will find an example log of OpenBeacon packets and false positives at <http://pastebin.com/8CbXHzJ9>. Sorting the list reveals that the MAC address 0x0102030201 is the most popular, which is in fact the address used by OpenBeacon tags.



Rather than rely on packet dumps and sorts, there is an autotune script that identifies network participants and prints their MAC addresses. Simply run 'goodfet.nrf autotune | tee autotune.txt' and go out for a coffee break while your device is transmitting. When you come back, you'll find logs like the following, which has identified a nearby OpenBeacon transmitter.

```
air-2% goodfet.nrf autotune
Autotuning as 0000000055 on 2499 MHz
sync,mac,r5,r6
Tuned to 2480 MHz
Tuned to 2481 MHz
'55,0102030201,51,09' looks valid      1      0.00820
'55,0102030201,51,09' looks valid      2      0.01600
'55,0102030201,51,09' looks valid      3      0.02326
'55,0102030201,51,09' looks valid      4      0.02837
Tuned to 2482 MHz
Tuned to 2483 MHz
```

As low data-rate devices require significantly more time than high-rate devices to identify, such devices will either require undue amounts of patience or a real KeyKeriki. In the case of a Nike+ foot pod, I'm resorting to using loud hip hop music to trigger the sensor, which is left inside a pair of headphones. My labmates are not amused, but it is a great way to reveal the radio settings when syringe probes aren't convenient.

### Part 3: or, Sniffing a keyboard effectively.

Having a class to identify channels and MAC addresses is most of the problem, but there are remaining issues. First, the packets themselves are encrypted, and that cryptography must be broken.

Fear not! We won't need to do any fancy math to break this cryptography, as the key is included at least once in every packet. Moser and Schröder's slides explain that the packet's header is cleartext, while the payload is XOR encrypted with the MAC address.

# Kiss your security goodbye

<b>C</b>	0A	78	06	01	<b>C2</b>	<b>98</b>	76	0A	C0	<b>C8</b>	98	35	0A	C0	C
<b>K</b>					<b>CD</b>	<b>98</b>	<b>35</b>	<b>0A</b>	<b>C0</b>	<b>CD</b>	98	35	0A	C0	C
<b>P</b>	0A	78	06	01	<b>0F</b>	<b>00</b>	<b>43</b>	<b>00</b>	<b>00</b>	<b>05</b>	<b>00</b>	<b>00</b>	<b>00</b>	<b>00</b>	<b>0</b>
	Dev Pac	ice ket	Mod	?	<b>Sequen</b>	<b>Flags/</b>				<b>HID</b>					
	typ typ	el			<b>ce ID</b>	<b>Meta</b>				<b>Code</b>					
	e	e													

(Key-Down) Packet with device address:  
**CD 98 35 0A C0**



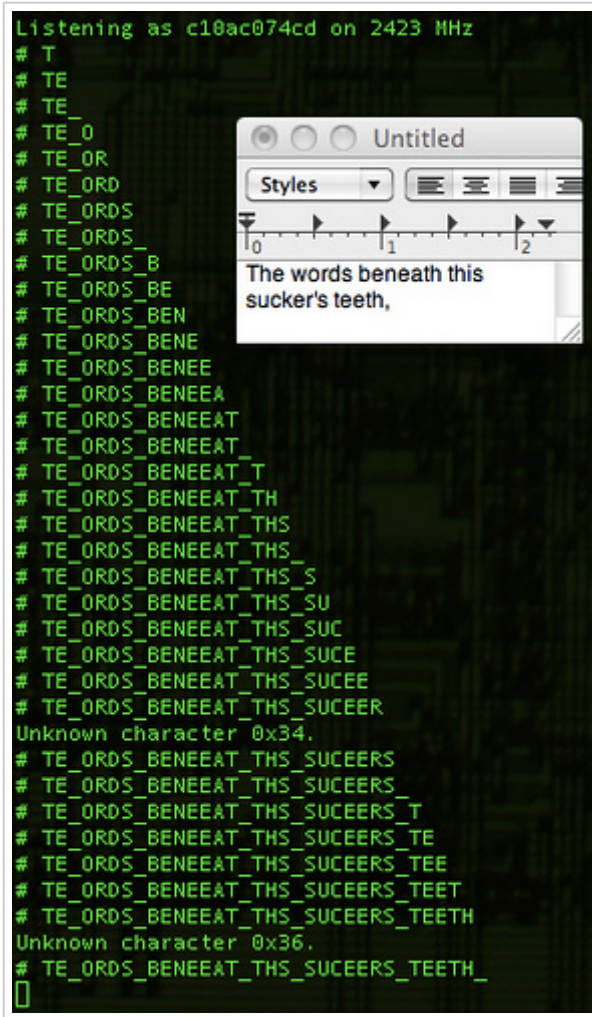
digital v00d00 - 8th of Decem  
Thorsten Schröder, Max Moser

Applying an XOR to the proper region yields decrypted packets such as the following. Because these contain USB HID events, key-up HID events quite often include long strings of 0x00 bytes. When XOR'ed with the key, those zeroes produce the key, so some packets contain the XOR key not just once, but twice!

```

dsl-dhcp-121% goodfiet.nrf sniffaskb aa.c18ac074cd,17.00 | tee ~/Desktop/mskb_l
Listening as c18ac074cd on 2423 MHz
0a 78 09 01 92 00 43 00 00 11 00 00 00 00 00 45 f1 3b f0 fc 45 f1 3b f0 fc 45
0a 38 09 01 92 00 00 57 9c 90 29 9d 57 9c 90 29 9d 57 9c 90 29 9d 57 9c 90 29
0a 38 09 01 92 00 00 57 9c 90 29 9d 57 9c 90 29 9d 57 9c 90 29 9d 57 9c 90 29
0a 78 09 01 93 00 43 00 00 00 00 00 00 00 00 55 e1 2b e0 ec 55 e1 2b e0 ec 55
0a 78 09 01 94 00 43 00 00 05 00 00 00 00 00 57 e3 29 e2 ee 57 e3 29 e2 ee 57
0a 38 09 01 94 00 00 51 cb f5 7d c1 53 b5 0d 2f 9b 51 9a 96 2f 9b 51 9a 96 2f
0a 38 09 01 94 00 00 51 cb f5 7d c1 53 b5 0d 2f 9b 51 9a 96 2f 9b 51 9a 96 2f
0a 78 09 01 95 00 43 00 00 00 00 00 00 00 00 53 ca 72 c8 cc 2e b4 89 cb 0c b9
0a 78 09 01 96 00 43 00 00 19 00 00 00 00 00 49 fd 37 fc f0 49 fd 37 fc f0 49
0a 78 09 01 97 00 43 00 00 19 06 00 00 00 00 4e ca 72 c8 cc 2e b4 89 cb 0c a0
0a 78 09 01 98 00 43 00 00 19 00 00 00 00 00 47 ca 72 c8 cc 2e b4 89 cb 0c a0
0a 38 09 01 98 00 00 5d cb f5 7d c1 5f b5 0d 23 ca 72 c8 cc 2e b4 89 cb c7 7e
0a 38 09 01 98 00 00 5d 96 9a 23 97 5d 96 9a 23 97 5d 96 9a 23 97 5d 96 9a 23
0a 78 09 01 9a 00 43 00 00 05 00 3f f5 3f bf 0e ca 72 c8 cc 23 b4 89 cb 0c bc
0a 38 09 01 9a 00 00 5f 94 98 21 95 5f 94 98 21 95 5f 94 98 21 95 5f 94 98 21
0a 78 09 01 9b 00 43 00 00 00 00 00 00 00 00 5d e9 23 e8 e4 5d e9 23 e8 e4 5d
    
```

Finally, the USB HID events need to be deciphered to get key positions. Mapping a few of these yields meaningful text, with bytes duplicated in the case of retransmissions and omitted in the case of lost packets. Disabling checksums will allow the dropped packets to be converted to a smaller number of byte errors, while tracking sequence numbers will prevent retransmitted keys from being displayed twice. Regardless, the results are quite neighborly, as you can make out the sentence typed below in its packet capture.



```

Listening as c10ac074cd on 2423 MHz
# T
# TE
# TE_
# TE_0
# TE_OR
# TE_ORD
# TE_ORDS
# TE_ORDS_
# TE_ORDS_B
# TE_ORDS_BE
# TE_ORDS_BEN
# TE_ORDS_BENE
# TE_ORDS_BENEE
# TE_ORDS_BENEEA
# TE_ORDS_BENEEAT
# TE_ORDS_BENEEAT_
# TE_ORDS_BENEEAT_T
# TE_ORDS_BENEEAT_TH
# TE_ORDS_BENEEAT_THS
# TE_ORDS_BENEEAT_THS_
# TE_ORDS_BENEEAT_THS_S
# TE_ORDS_BENEEAT_THS_SU
# TE_ORDS_BENEEAT_THS_SUC
# TE_ORDS_BENEEAT_THS_SUCE
# TE_ORDS_BENEEAT_THS_SUCEE
# TE_ORDS_BENEEAT_THS_SUCEER
Unknown character 0x34.
# TE_ORDS_BENEEAT_THS_SUCEERS
# TE_ORDS_BENEEAT_THS_SUCEERS_
# TE_ORDS_BENEEAT_THS_SUCEERS_T
# TE_ORDS_BENEEAT_THS_SUCEERS_TE
# TE_ORDS_BENEEAT_THS_SUCEERS_TEE
# TE_ORDS_BENEEAT_THS_SUCEERS_TEET
# TE_ORDS_BENEEAT_THS_SUCEERS_TEETH
Unknown character 0x36.
# TE_ORDS_BENEEAT_THS_SUCEERS_TEETH_

```

#### Part 4; or, Reproducing these results.

All of the code for this article is available in the GoodFET Project's repository, as part of GoodFETNRF.py and its goodfet.nrf client script. The hardware used was an NHBadge12, although an NHBadge12B or a GoodFET with the SparkFun nRF24L01+ Transceiver Module will work just as well.

To identify a nearby Nordic transmitter, run 'goodfet.nrf autotune'. Keyboards can be identified and sniffed with 'goodfet.nrf sniffmskb', while a known keyboard can be sniffed and decoded by providing its address as an argument, 'goodfet.nrf sniffmskb aa,c10ac074cd,17,09'. The channel--0x17 in this case--will change for collision avoidance, but channel hopping is slow and resets to the same starting channel. Identification of the broadcast channel is faster when the receiver is not plugged in, as that causes the keyboard to continuously rebroadcast a keypress for a few seconds.

All code presently in the repository will be refactored and rewritten, so revert to revision 885 or check the documentation for any changes.

#### Conclusions

Contrary to prior belief, the nRF24L01+ *can* be used to promiscuously sniff compatible radios, allowing for keyboard sniffing without special hardware. It's also handy for figuring out the lower levels of the otherwise-documented ANT+ protocol, and for reverse engineering vendor-proprietary protocols such as Nike+.

Additionally, it should be emphasized that the security of the Microsoft keyboards in this family is irreparably broken, and has been since Moser and Schröder published the vulnerability at CanSecWest. (It's a shame, because the keyboards are quite nicer than most Bluetooth ones, both in pairing delay and in battery life.) Do not purchase these things unless you want to broadcast every keystroke.

While I have not yet written code for injecting new keystrokes, such code does exist in the KeyKeriki repository and would not be difficult to port. Perhaps it would be fun to build stand-alone firmware for the Next Hope badge that sniffs for keyboards, broadcasting Rick Astley lyrics into any that it finds?

Please, for the love of the gods, use proper cryptography and double-check the security your designs. Then triple-check them. There is no excuse for such vulnerable garbage as these keyboards to be sold with neither decent security nor a word of warning.

POSTED BY TRAVIS GOODSPEED AT 1:09 AM  
LABELS: KEYKERIKI, MICROSOFT, NORDIC, NRF24L01+

---

## 8 COMMENTS:



XTL said...

I love it.

FEBRUARY 7, 2011 AT 6:55 AM



Nelson Lombardo said...

brootal!

FEBRUARY 7, 2011 AT 2:10 PM



Anushka Sharma said...

nice article... simple and useful :).... Plz visit my site...! Thanks  
Technical Support Engineering Services in Delhi,

AUGUST 21, 2015 AT 6:36 AM



\_ Яafild ☺ \_ said...

fantastic

OCTOBER 17, 2015 AT 10:34 AM