

# Hi, My Name is Keyboard

Marc Newlin @ ShmooCon 2024

# What is this talk about?

- Bluetooth link-key extraction vulnerabilities
  - macOS (via USB)
  - Magic Keyboard (via Lightning and Bluetooth)
- Bluetooth keystroke-injection vulnerabilities
  - macOS
  - iOS
  - Android
  - Linux (BlueZ)
  - Windows

# Marc Newlin

Principal Reverse Engineer @ SkySafe





# Gaming Keyboards

I wanted to do some stunt-hacking, and gaming keyboards looked fun

- proprietary wireless protocols
- addressable RGB LEDs
- flagship peripherals probably have good security (lol)

# Alienware AW920K

- nRF52 SoCs in the keyboard and dongle have SWD disabled
- no firmware updates were available when I was exploring this keyboard
- pointed a USB control-transfer fuzzer at the dongle
- dongle ~bricked after 30-seconds of fuzzing
- SWD was enabled after plug-cycling the dongle (lol)
- turns out two pages of bootloader flash had been erased (wat)

# Alienware AW920K

- RF comms are encrypted using AES ECB
  - 16-byte AES key is stored above the application in flash
  - link key is composed of 11-bytes from the above key, and the 5-byte RF address of the dongle
  - I suspect the key is static, but have not verified with another dongle
- Firmware updates are now available as Windows apps which contain encrypted firmware images
  - You can extract and decrypt them with this script:  
<https://gist.github.com/marcnewlin/c936d1530fb5e3648aca23d7c08c9628>

# It's AppleTooth time!

"Bluetooth is probably secure because it uses fancy encryption."

"Apple keyboards are probably secure because Apple."

-- me before starting this research



# Enter the Magic Keyboard A2450



**But first, a word about Bluetooth and HID.**

# Bluetooth Classic != Bluetooth Low Energy

Bluetooth keyboards can implement HID over BT or BLE

- **Bluetooth Classic** keyboards use the Human Interface Device Profile (**HIDP**)
- **Bluetooth Low Energy** keyboards use the HID over GATT Profile (**HOGP**)

The vulnerabilities in this talk are specific to Bluetooth Classic / HIDP

# Bluetooth Classic L2CAP Sockets

The L2CAP layer provides socket connections between Bluetooth Classic hosts and peripherals

```
import bluetooth

bt_addr = "01:23:45:67:89:ab"
port = 1

sock = bluetooth.BluetoothSocket(bluetooth.L2CAP)
sock.open((bt_addr, port))
```

# Bluetooth Profiles

Bluetooth **Profiles** are APIs that define the various services a Bluetooth device can implement

- 80+ profiles are defined in the Bluetooth specification
- common Bluetooth stacks support on the order of 10-20 host profiles
- most phones and computers support at least one of the Bluetooth HID profiles

# Service Discovery

Bluetooth devices advertise their supported profiles using the Service Discovery Protocol

- when a device is **discoverable**, other Bluetooth devices can connect to SDP and enumerate the supported profiles
- SDP is an unauthenticated and runs on L2CAP port 1
- service definitions include L2CAP port numbers and profile-specific configuration

# HIDP L2CAP Services

The Human Interface Device Profile uses two L2CAP services

- **HID Control** on port 17
  - synchronous channel used for configuration
- **HID Interrupt** on port 19
  - asynchrnous channel used for input events and state changes

# HID Reports

Human Interface Devices communicate by sending and receiving messages called **reports** over USB or Bluetooth

- **Input** reports include keypresses and mouse movement/clicks
- **Output** reports include commands and state changes
- **Feature** reports are used by the host to read and write device settings



# Bluetooth HID Connection Establishment

A **paired** keyboard can connect to L2CAP 17 and 19 and send input reports.

```
import bluetooth

host = "01:23:45:67:89:ab"

control = bluetooth.BluetoothSocket(bluetooth.L2CAP)
interrupt = bluetooth.BluetoothSocket(bluetooth.L2CAP)

control.open((host, 17))
interrupt.open((host, 19))

# send 'a' keypress
interrupt.send(b"\xa1\x01\x00\x00\x04\x00\x00\x00\x00\x00\x00")
```

# Bluetooth Pairing Concepts

- **Pairing** establishes the link key which is used to encrypt the data sent between two Bluetooth devices
- **Bonding** saves the link key to the device
- **NoInputNoOutput** pairing is used by devices which do not support authentication
- **Out of Band Pairing** performs pairing and bonding over a non-Bluetooth channel like NFC or USB

**Okay, so how hack Apple?**

# Magic Keyboard Fuzzing

1. Point a USB control-transfer fuzzer at the Apple Magic Keyboard
2. ...
3. Bricked?

```
[ +0.479931] usb 1-2: new full-speed USB device number 40 using xhci_hcd
[ +0.020590] usb 1-2: device descriptor read/8, error -61
[ +0.128398] usb 1-2: device descriptor read/8, error -61
[ +0.303034] usb 1-2: new full-speed USB device number 41 using xhci_hcd
[ +0.020882] usb 1-2: device descriptor read/8, error -61
[ +0.128111] usb 1-2: device descriptor read/8, error -61
[ +0.107077] usb usb1-port2: unable to enumerate USB device
```

# Magic Keyboard Write Primitives

The fuzzer soft-bricked the keyboard by writing an empty USB device descriptor, but it reverts to the default value when the keyboard is switched off and back on

## USB Descriptors

- product name, serial number, and device descriptor

## HID Feature Reports

- product name, serial number, and Bluetooth address

# Magic Pairing?

Magic Keyboard pairs with the Mac over USB

Does pairing change any USB descriptors or HID reports?

1. Dump USB Descriptors and HID Feature Reports to `before.txt`
2. Plug the Magic Keyboard into a Mac and then unplug it
3. Dump USB Descriptors and HID Feature Reports to `after.txt`

# HID Feature Report 0x35

```
$ diff before.txt after.txt  
< [HID 0x35] 350000000000000000000000000000000000000000000000000000000  
---  
> [HID 0x35] 350101a4cf9988086dd8a5a142cc5d34c17383329dbce27d96
```

# Uhhhhhhh

# Bluetooth address of the Mac

A4CF9988086D

# 16 bytes of high-entropy data (!!!)

D8A5A142CC5D34C17383329DBCD27D96





# Magic Keyboard USB Pairing

The first time a Magic Keyboard is plugged into a Mac, `bluetoothd` generates a random link key and sends it to the keyboard over USB

```
bluetoothd: Found USB Device
bluetoothd: Successfully got device report
bluetoothd: BT ADDR for device name:
bluetoothd: Successfully generated Link Key preparing it to send to device
bluetoothd: Successfully resent Link Key to paired device
bluetoothd: Attempting To Pair device
bluetoothd: SUCCESSFULLY PAIRED ADDR: 1C:57:DC:88:55:02
bluetoothd: handleIncomingUSBDevice deviceId information: vidSrc 0x4c ...
bluetoothd: Adding HID device with location ID 1048576
bluetoothd: Adding USB paired device to SFLASH
```

# HID Report 0x35 == Out-of-Band Pairing Data

```
350101A4CF9988086DD8A5A142CC5D34C17383329DBCD27D96
```

```
# Report ID (and some sort of header?)  
350101
```

```
# Bluetooth address of the Mac  
A4CF9988086D
```

```
# Bluetooth Link Key  
D8A5A142CC5D34C17383329DBCD27D96
```

# Magic Keyboard USB Pairing (second connection)

The next time the Magic Keyboard is plugged into the Mac, the **same** link key is sent to the keyboard over USB

```
bluetoothd: Found USB Device
bluetoothd: Successfully got device report
bluetoothd: BT ADDR <private> for device name:
bluetoothd: Device Already paired
bluetoothd: Adding HID device <private> with location ID 1048576
bluetoothd: Seeing if paired device Link Key already exists for iohid ...
bluetoothd: Found HID device <private> with location ID 1048576
bluetoothd: Preparing to send existing Link Key to <private>
bluetoothd: Successfully resent Link Key to paired device
```

# Read the Magic Keyboard Link Key (Lightning)

- After the Magic Keyboard is plugged into the Mac, the link key remains in memory until the keyboard is switched off
- An attacker can read the link key by plugging into the keyboard's Lightning port and reading HID report 0x35
- The link key can be used to inject keystrokes into the Mac, or to MITM the link and sniff keystrokes

# Unauthenticated Bluetooth HID

## 5.4.3.3.3 Backwards Compatibility

The following are requirements and recommendations for Bluetooth HID devices when interoperating with Bluetooth HID Hosts that are compliant to Bluetooth Core Specification versions prior to v2.1+EDR:

---

**Bluetooth HID devices in Discoverable Mode should allow the HID Control channel and HID Interrupt Channel to be opened without encryption.**

# Magic Keyboard USB-to-Bluetooth Handoff

- While connected over USB, the Bluetooth radio is powered off
- When unplugged
  - the Bluetooth radio on the Magic Keyboard turns on and runs unauthenticated HIDP until the Bluetooth link is established with the Mac
  - the Mac connects to HIDP on the Magic Keyboard

# SDP Unplug Trigger

- when the Magic Keyboard is unplugged from the Mac, the Mac connects to SDP on the keyboard
- if the attacker is spoofing the keyboard and is connected to SDP on the Mac, the Mac will connect to SDP on the attacker's device when the keyboard is unplugged

# Read the Magic Keyboard Link Key (Bluetooth)

- an attacker can use the SDP trigger to detect the unplug event and connect to HIDP on the Magic Keyboard
- the link key can then be read from HID report 0x35
- **exposing credentials over unauthenticated channels is suboptimal**



# Bluetooth Super Server Robot Destroyer

When `bluetoothd` launches on macOS, it identifies itself in the logs as `Bluetooth Super Server Robot Destroyer`

- clearly this is a formiddable opponent
- may the hacker gods be on our side

So how does the robot destroyer authenticate the Magic Keyboard?

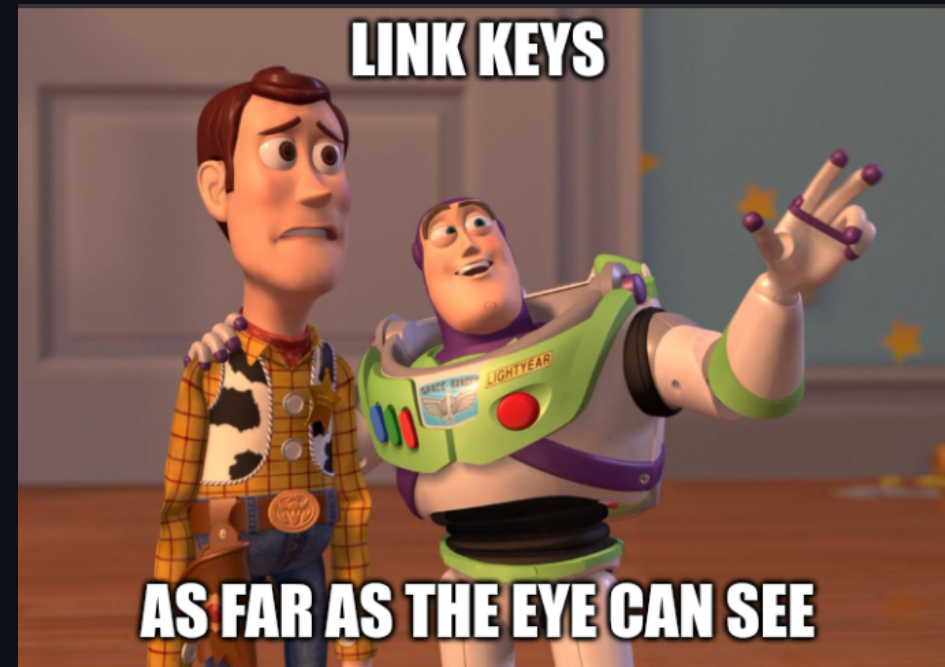
... using its **Bluetooth address** and **serial number**

# Reading the Link Key from a Screen-Locked Mac

1. Write the Bluetooth address and serial number of the target keyboard to an donor keyboard
2. Plug the donor keyboard into the target Mac for a few seconds
3. Read HID report 0x35 from the donor keyboard

# Magic Keyboard Link Key Extraction

- from the Magic Keyboard over **Lightning**
- from the Magic Keyboard over **Bluetooth**
- from the Mac over **USB**



# Pairing a Magic Keyboard to a Malicious Host

Out-of-band pairing data can be written over Lightning or Bluetooth

1. Attacker writes OOB pairing data to a Magic Keyboard over Lightning or Bluetooth
2. The Magic Keyboard then connects to the attacker-defined host using the attacker-defined link key

# Machine-in-the-Middle

- We can connect to the Magic Keyboard without authentication when the Bluetooth radio comes online
- If we know the link key, we can spoof the keyboard and connect to the Mac over Bluetooth

# macOS Forced Pairing

- When the Mac is attempting to connect to a Magic Keyboard, an attacker can spoof the keyboard and pair with the Mac without authentication
- Using the SDP connection as a timing trigger, this attack works with high reliability
- macOS 12 and 13 remain vulnerable
- macOS 14 was patched in 14.2







# What is even happening?

Bluetooth HID devices can pair without authentication using the `NoInputNoOutput` pairing capability

Hosts are vulnerable to this attack under the following conditions:

- host supports `NoInputNoOutput` pairing
- host does not prompt for user confirmation when using `NoInputNoOutput`
- host allows a Bluetooth keyboard to initiate pairing



# iOS Forced Pairing

- iOS pairs with Magic Keyboard over Bluetooth, so no unplug event
- attack similar to macOS is possible when the iPhone reconnects to its paired Magic Keyboard
- SDP trigger functions the same as during the unplug event on macOS
- iOS 16 is still vulnerable, iOS 17 was patched in 17.2

C

▶ 0:00 / 0:12



# What about Android?

- iOS attack reproduced on Android
- only I messed up the PoC and was spoofing the wrong keyboard
- the Android attack has **no spoofing component**
- attacker can force-pair a Bluetooth keyboard with Android **anytime Bluetooth is enabled**, without user confirmation (!!!)
- Android 4.2.2 - 14 is vulnerable, fixed in 2023-12-05 security patch level

# Maybe Linux wants to join the party?

- attacker can force-pair a Bluetooth keyboard without user confirmation **anytime the Linux host is discoverable and connectable**
- the Linux/BlueZ bug was fixed in 2020, but ChromeOS was the only Linux distro that enabled the fix (oops)
- Linux with BlueZ 5.x is vulnerable prior to patches released in December 2023



Settings

Bluetooth

Network

Bluetooth

Background

Appearance

Notifications

Search

Multitasking

Applications

Privacy

Online Accounts

Bluetooth


Visible as "test-vm" and available for Bluetooth file transfers. Transferred files are placed in the [Downloads](#) folder.

Devices

[Redacted]	Not Set Up
[Redacted]	Not Set Up
[Redacted]	Not Set Up
[Redacted]	Not Set Up
[Redacted]	Not Set Up
[Redacted]	Not Set Up



Home

▼ ⚙ 2 ■■■■■ profiles/input/input.conf 



@@ -17,7 +17,7 @@

17	17	# platforms may want to make sure that input connections only come from bonded
18	18	# device connections. Several older mice have been known for not supporting
19	19	# pairing/encryption.
20		- # Defaults to false to maximize device compatibility.
	20	+ # Defaults to true for security.
21	21	#ClassicBondedOnly=true
22	22	
23	23	# LE upgrade security



# Windows Forced Pairing

- attacker connects to the Windows host while spoofing an already-paired keyboard
- a pairing request is presented to the user
- if the user interacts with the pairing request in any way (accept, reject, or close), the attacker can pair without authentication and inject keystrokes
- Windows 10, 11 and Server 2022 were patched by Microsoft on 2023-01-09

# Demo



# What does Bluetooth SIG think about this?

"At the surface level, the description does appear to be implementation-specific, even if multiple implementations are independently vulnerable. Though the HID specification does permit implementations to use non-MITM-protected association models for HID services, **this is not an endorsement of their use, merely the absence of a prohibition against their use**, as there are use cases where the MITM-mitigating association models would not be supported (e.g., a Bluetooth mouse)."

-- Bluetooth SIG

# Disclosure

- I disclosed the bugs in August and September of 2023
- First to Apple and Google, then Canonical and Bluetooth SIG, then Microsoft
- Android, Linux, macOS and iOS fixes landed in December 2023
- Windows fix landed in January 2024
- Apple landed fixes for the link-key bugs in January 2024
- Big thanks to Marc Esler of Canonical for helping wrangle the Linux/BlueZ disclosure, and to all the vendors for fixing in time for ShmooCon :)

# Aftermath

- Fixing the Android bug broke Fast Pair and Nearby Share
- Fixing the Linux bug broke support for certain peripherals (like the PS3 controller)
- My Bluetooth headphones no longer work with Linux

# Retrospective

- The vendors dropped the ball by missing these bugs
- **We dropped the ball** by not finding them sooner
- We are all human and humans make mistakes
- Hilarious bugs are hiding in plain sight, waiting for you to find them

# Happy Hacking :)

proof-of-concept code will go live immediately following this talk

[https://github.com/marcnewlin/hi\\_my\\_name\\_is\\_keyboard](https://github.com/marcnewlin/hi_my_name_is_keyboard)

- keystroke injection in Android, Linux, macOS, iOS and Windows
- link key extraction from Magic Keyboard over Lightning and Bluetooth
- link key extraction from Mac over USB

@marcnewlin on GitHub and Twitter