The Impostor Among US(B): Off-Path Injection Attacks on USB Communications

Robert Dumitru, *The University of Adelaide and Defence Science and Technology Group;* Daniel Genkin, *Georgia Tech;* Andrew Wabnitz, *Defence Science and Technology Group;* Yuval Yarom, *The University of Adelaide*

https://www.usenix.org/conference/usenixsecurity23/presentation/dumitru

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

USENIX'23 Artefact Appendix: The Impostor Among US(B): Off-Path Injection Attacks on USB Communications

Robert Dumitru
The University of Adelaide &
Defence Science and Technology Group
robert.dumitru@adelaide.edu.au

Daniel Genkin
Georgia Institute of Technology
genkin@gatech.edu

Andrew Wabnitz
Defence Science and Technology Group
andrew.wabnitz1@defence.gov.au

Yuval Yarom
The University of Adelaide
yval@cs.adelaide.edu.au

A Artefact Appendix

A.1 Abstract

The artefact is an instance of a USB device capable of injecting transmissions that a USB host will attribute to a neighbouring connected device, as described in the paper. It is configured to present itself as a USB mouse and can inject keystrokes on behalf of an adjacently connected USB keyboard, while optionally also blocking genuine input from the keyboard victim.

The artefact is in the form of a bitstream to be programmed onto an FPGA training board (ported for a Basys 3). The RTL source is also provided for modification and re-generation of the bitstream for use on other boards. A 1.5 kΩ resistor and basic cable splicing/rewiring is required.

With reference to the paper, the basis of this artefact is described in Section 5.1, and the claims to be reproduced are described in Section 7.1.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There is no risk of destructive behaviour from evaluation of this artefact.

A.2.2 How to access

Most recent source available at:
https://github.com/0xADE1A1DE/USB-Injection/

Stable tag:
https://github.com/0xADE1A1DE/USB-Injection/releases/tag/PosSec23AE

A.2.3 Hardware dependencies

Target hardware: A Digilent Basys 3 FPGA development board is required. This is a cheap (149 USD) and commonly used FPGA board. If you are at a university with an Elec Eng department, they are likely to have some of these available. (Other FPGA boards can be used if they have 3.3V IO, modification of constraints file and generation of a new bitstream from RTL source would be required).

Supplementary:
- A USB A to Micro B cable is needed to program the FPGA board (this should come in the Basys 3 box)
- A secondary USB cable must be spliced (exposing internal connector wires) while leaving the side of the type-A male connector intact (part that plugs into computer USB ports)
- 1.5 kΩ resistor
- Wires / connectors / breadboard

For testing:
- Any PC with USB ports
- A LS (Low-Speed) keyboard (majority are LS)
- USB hub(s)

A.2.4 Software dependencies

Any version of Xilinx's Vivado software, including free versions, can be used to configure the injection platform. See https://www.xilinx.com/support/download.html for latest versions.

[Windows users] We highly recommend USB Device Tree Viewer (https://www.uwe-sieber.de/usbtreeview_e.html) by Uwe Sieber for viewing the complete hierarchy of USB devices connected to your computer along with their descriptor sets. This will help to confirm the device is working.

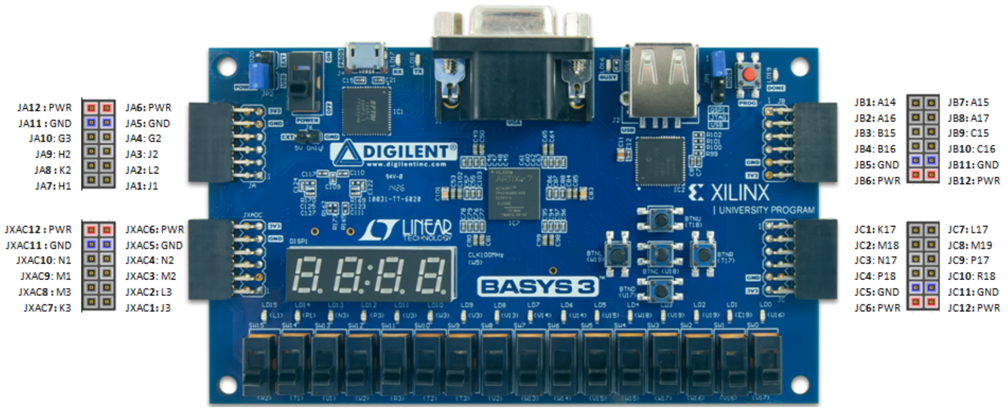

Figure 1: Basys 3 Pmod pinout

A.3 Set-up

A.3.1 Installation

1. Connect Basys 3 board to a computer running Vivado and program the target FPGA with the bitstream file from the repository (OS is according to the host computer the injection platform will be plugged into – this can be the same PC to which the Basys 3 is connected for programming):
   [*Windows*]
   `LS Keystroke Injector > USB_Demo.bit`
   [*Linux*]
   `LS Keystroke Injector > USB_Demo_Linux.bit`
2. Connect wires from a spliced USB cable to the Basys 3 board with the pin correspondence described in Table 1. See Figure 1 and Figure 2.

USB pin	USB wire colour	Basys 3 JB Pmod pin
D+	Green	JB1
D-	White	JB3
Gnd	Black	JB5
Vs	Red	Leave unconnected

Table 1: USB wire to FPGA pin correspondence

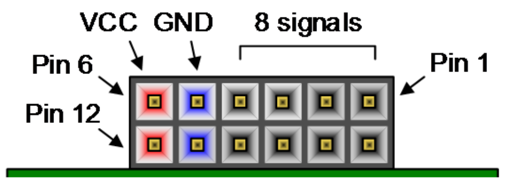

Figure 2: Pmod connector header pin numbering

3. Pull up the D- line to 3.3V across a 1.5kΩ resistor (as in Figure 3). To do this, you can connect one side of the resistor through JB6 (Vcc at 3.3V) on the same Basys 3 Pmod header, and connect the other side to the junction of JB3 and D- from the spliced cable.

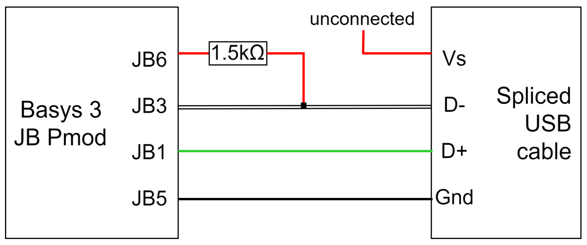

Figure 3: D- pullup resistor wiring

4. Continue to 'Basic Test'

A.3.2 Basic Test

With the Basys board programmed and connected to the spliced cable as instructed, plug the USB connector from the spliced cable into a PC (this can be the same PC to which the Basys 3 is connected for programming its FPGA). This should connect as a new mouse device. Confirm artefact functional as follows:
[*Windows*] Can either use device manager or preferably the USB Device Tree Viewer software previously mentioned in Appendix A.2.4.
[*Linux*] the `lsusb` command can be used to display the entire USB connection hierarchy.
Check the connection hierarchy with the device unconnected vs plugged in to confirm it appears.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The injector device can inject keystroke data on behalf of an adjacently connected victim keyboard when both the injector and victim are connected to the same vulnerable single-TT standard hub. This is proven by experiment (E1) described in Section 7.1 of the paper.

(C2): The injector device can effect a Denial-of-Service to the adjacently connected victim keyboard when both the injector and victim are connected to the same non-vulnerable, single-TT standard hub. This is also proven as an alternative outcome of experiment (E1) described in Section 7.1 of the paper.

A.4.2 Experiments

(E1): Keystroke injection *[10 human-minutes]:*

How to: Connect both the injection platform and victim keyboard to a common USB hub which is connected to a host PC. This can be the same host machine that the Basys board is connected (or any other machine). Connect no other devices through the hub. The hub is the device under test here, if it is vulnerable to injection the injection platform will be able to inject keystroke data on behalf of the connected victim keyboard [**Results**].
If the hub is not vulnerable, the injector should still be able to effect a Denial-of-Service against the victim keyboard [**Alternative Results**].

Preparation: Ensure the injection platform and keyboard are both logically connected to the same USB 2.0 hub and are both operating at LS. This will require using USB Device Tree Viewer (Windows) or the `lsusb` command (Linux).
The common hub must not be the computer root hub, injection will not work against root hubs.
Ensure the USB 2.0/2.1 hub is single-TT.
Ensure the Reset switch (SW0) is off (down), this is furthest right of the switches lining the bottom of the board (Figure 1).
Configure board switches into State 3 as in Table 2.

State	inj (SW1)	DoS (SW2)	Behaviour
0	0	0	NAKs being injected
1	0	1	No injection - victim works
2	1	0	NAKs being injected
3	1	1	Data being injected

Table 2: Injection platform switch configurations

[*Linux*] Open a document in a text editor.
[*Windows*] Same as above or do nothing.

Execution: Push the buttons on the Basys 3 board (5 buttons arranged in + shape on the bottom right side of the board) as follows (orientation as shown in Figure 1):
[*Windows*] **Left**, **Left + Right**, release both, **Up**, release, **Centre**, release, **Down**, release, **Right**
[*Linux*] Push and release any of **Up**, **Centre**, **Down**, or **Right**.

Results: If the hub under test is vulnerable to injection, you should see the following behaviour:
[*Windows*] The sequence of injected keystrokes opens a command prompt.
[*Linux*] The following keys are typed on the text editor document: c (**Up**), m (**Centre**), d (**Down**), and enter (newline) (**Right**).
If possible try with various hubs. USB 2.0 hubs are likely to be vulnerable, whereas USB 3.0 hubs are unlikely as a very small portion have been found vulnerable.

Additional Evidence of Results: *[Optional – if injection working and hub vulnerable]*
Wireshark's USBPcap function can be used to view injected traffic.
Unplugging the victim keyboard and pressing the same buttons on the still-connected injector will result in no keystrokes being fed.
Install any software-based USB authorisation policies and attempt injection with the victim keyboard allowed while the injector is blocked – as in Section 8. Injection will still work.

Alternative Results: Denial-of-Service against the keyboard victim should still be evident with hubs that are not vulnerable to injection. With the injector connected, open a text document and attempt to type keystrokes through the victim keyboard. No keystrokes should pass through. Change the injector switches to State 1 and the victim should then be able to type keystrokes.

A.5 Notes on Reusability

We have made the source RTL available so the injector device can be modified with generation of new bitstreams. Note, this source is for the Windows-compatible injector. Compatibility issue is just a bug from how different OS drivers process descriptors which we have not yet resolved. Some possible alterations for reuse are as follows:
Changing injected input. To change what data is injected, modify what is written to `PCIn` in `USBF_Demo.vhd`. See HID keyboard scan codes for data corresponding to keystrokes.
Changing injector device descriptors. To change any of the descriptor fields (ID, device type, etc.), modify values in `USBF_Declares.vhd`. Injection function is agnostic to the injector platform device type.
Use injector as FS device to target gaming keyboards. Repeat experiment with files under `FS Keystroke Injector` and pull up D+ instead of D-.
Use with different boards. The injector configuration should work with various FPGA boards, all that is needed is 3.3V IO and a different set of constraints (`.xdc` file).

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220912. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenix%20sec2023/`.