Implementing USB Attacks with Microcontrollers

Alejandro Rodríguez Ocasio
Master in Computer Science
Advisor: Jeffrey Duffany, Ph.D.
Electrical and Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

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. In this article we will discuss different types of USB attacks and explore how tools capable of automating such attacks can be implemented by using a low-cost reprogrammable microcontroller.

Key Terms *— Bash Bunny, Human Interface Device (HID) Attacks, Penetration Testing Tools, Raspberry Pi, Universal Serial Bus (USB) Attacks.*

INTRODUCTION

As technology advances, a wider variety of digital devices are being constantly introduced. Most of these require some level of interaction by either a user or another device. Typical interactions include communication between a host PC and peripherals such as keyboards, mice, flash storage, and ethernet adapters. A variety of interfaces have existed in the past to facilitate interaction between devices and peripherals. Some of the most common included parallel ports, serial ports, PS/2 connectors, DIN connectors, and game ports among many others. Some of these were proprietary and most required dedicated expansion cards and specific configurations. This could become cumbersome, as host devices were required to support a wide variety of interfaces to increase compatibility with peripherals.

The introduction of the Universal Serial Port (USB) specification published in 1996 helped mitigate compatibility and configuration issues. Its main objective is to improve and simplify the interface between devices and peripherals. The USB specification establishes standard cables, connectors, and protocols. And it supports the management of up to 127 devices on a host. These devices are considered Plug and Play (PnP). This means that the configuration of the devices occurs automatically when plugged into a host and typically do not require any interaction with a user. The standard has gained widespread adoption, replacing a variety of older interfaces. USB interfaces are present in most modern devices, providing a ubiquity that greatly improves compatibility and ease of use.

To facilitate 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.

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. Though, securing an infrastructure against USB attacks can be difficult and is often overlooked.

In this article we will discuss the different categories of USB attacks that can be performed

through malicious devices. We will also provide insight into how tools that exploit these vulnerabilities can be easily implemented using a combination of low-cost components and open-source software. Specifically, we will demonstrate how we can replicate most of the functionality of the Bash Bunny [1], a commercially available penetration testing tool for automating USB attacks.

USB PROTOCOL

USB interfaces facilitate communication between host systems and I/O devices and optionally serve the purpose of supplying the electricity to the device. USB devices rely on microcontrollers with embedded CPUs to control the interaction with the host. Occasionally, they include bootloaders that allow updating the device's firmware.

Devices can provide one or more functions (i.e. keyboard, video recorder, Ethernet adapter), each of which corresponds to a logical address on the bus known as endpoint. Each endpoint forms a pipe, which is a logical communication channel. Pipes are divided into two types: message and stream. Message pipes are used for control transfers, which consist of configuration and control information. Stream pipes are used for interrupt, bulk, or isochronous transfers. Interrupt transfers consist of small quantities of time-sensitive data., while bulk transfers consist of large quantities of time-insensitive data. Isochronous transfers consist of timing real-time data at predetermined transfer rates requiring timing coordination.

When a USB device is connected to a host, a process known as enumeration is initiated. This process consists of detecting the device, determining the device's speed, determining the device's descriptors, and loading the required drivers.

During the enumeration process, the host attempts to detect the device by monitoring voltage fluctuations on the data lines. Once detected, the host attempts to determine the device's speed. The device is then reset, and the speed of the host is matched to proceed to read the device's descriptors that identify the device. The device is reset again and given a unique logical address. The host will then proceed to retrieve configuration descriptors that include the interface and endpoint. With this information, the host determines and loads the associated driver that will allow it to control the device. Identification of the corresponding driver is typically performed by matching the device's USB class, vendor ID, and product ID.

USB ATTACKS

A study performed by Honeywell reveals that USB attacks have been and remain one of the main attack vectors in the industry [2]. Many attacks have been developed that aim to exploit the vulnerabilities of USB interfaces. Executing these attacks will typically consist of simply plugging the malicious device into a USB port of an unprotected system. Once this occurs, the possibilities available to the attacker are endless and the results can be devastating. These include data exfiltration and destruction, installation of backdoors, deactivation of services, and even deactivation or destruction of the host. Most of these attacks are hard to detect and are typically executed covertly without a user's knowledge.

Researchers from the Ben-Gurion University of the Negev in Israel have identified 29 different types of USB attacks [3][4]. They divided the attacks into four main categories, which we will proceed to describe.

Reprogrammable Microcontroller USB Device

Attacks in this category consist of USB devices with programmable microcontrollers that provide the capability of emulating different types USB peripherals. Typical attack vectors employed by these devices include Human Interface Device (HID) peripherals, ethernet adapters, serial devices, and storage devices. Once connected to the host, these behave as a regular USB peripheral but provide a level of control on the machine equivalent to that of a real user. For example, a

device posing as a HID keyboard has the capability of simulating a real user on a keyboard, while a device posing as an ethernet adapter has the capability of manipulating network traffic.

Devices under this category can be either bought or built for a relatively low cost. Popular commercially available products under this category include the Bash Bunny [1] and the Rubber Ducky [5], while do-it-yourself (DIY) alternatives include URFUKED [6], USBdriveby [7], TURNIPSCHOOL [8], and USBHarpoon [9]. Specifications for DIY devices are typically maintained by the open-source community. These usually rely on low-cost components, making use of commercially available programmable microprocessors such as Arduino [10], Teensy [11], and Raspberry Pi [12].

USB Peripheral with Reprogrammed Firmware

These attacks rely on reprogramming the drivers or firmware of a common USB peripherals so that they may execute malicious scripts when plugged into the host. Carrying out these attacks usually involves a high level of technical complexity, making their implementation more challenging than other types of attacks. On the other hand, a clear advantage offered by these attacks is being nearly undetectable as they do not require hardware modifications and the required software is concealed within the firmware of a seemingly normal peripheral. Devices under this category are commonly known BadUSB [13] and become reprogrammed for malicious purposes through infected hosts. Possible attacks with these devices include HID attacks, DNS overrides, password protection bypassing, and data exfiltration through hidden partitions.

Software Exploits through Unmodified USB Peripherals

Instead of depending on custom hardware or firmware, these attacks rely on exploiting software and USB protocol vulnerabilities through unmodified USB peripherals. Stuxnet [14] is a popular example of these attacks. It used a regular USB storage device to execute a script contained within the device by exploiting how Windows automatically managed .LNK files when a USB storage device is plugged. The Fanny malware, on the other hand, takes advantage of how operating systems handle data on USB storage devices by hiding it in seemingly corrupted sectors that are ignored by the system. Though not all attacks in this category originate from the device. The Device Firmware Update (DFU) exploit, for example, can be used to infect USB devices plugged into an infected host to convert them into a BadUSB [13].

USB Electrical Attack

USB electrical attacks take advantage of the lack of protection in USB power and data lines present in most devices. These attacks consist of devices composed of electrical hardware components which generate an electrical surge to cause irreparable damage to a host. One of the most popular commercially available devices in this category is the USB Killer [15]. When this device is connected to a host, it collects power from the USB power lines until it reaches -240V. It then proceeds to discharge it through the USB data lines, repeating the process until the host is destroyed.

DEVICE IMPLEMENTATION

We will be focusing on the implementation of a penetration testing tool that falls into the attack category of reprogrammable microcontroller USB devices discussed in the previous section. The USB device can replicate most of the functionality of the commercially available Bash Bunny [1] from Hak5. 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 [5], a commercial tool developed by Hak5 to perform HID attacks, can also be emulated in the device.

Like the Bash Bunny, the device is 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. Hak5 maintains libraries of payloads for the Bash Bunny [16] and the Rubber Ducky [17] that can easily be adapted to the device.

Our implementation will be based on Alex Jensen's article titled "Poor Man's Bash Bunny" [18]. 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

A variety of reprogrammable microcontrollers and components may be used to achieve the same results. Every hardware component required is commercially available at a relatively low-cost. The following section describes the hardware components specific to our implementation, while Figure 1 details how these components interact.

- **Raspberry Pi Zero W:** Compact low-cost single-board computer. Contains most of the hardware components required for implementing our penetration testing tool. Includes a 1GHz single core CPU, 512MB of RAM, an 802.11 b/g/n wireless LAN adapter, a USB interface, and the 40 I/O pins [19].
- **Micros SD card**: Main boot device for the Raspberry Pi. Contains the operating system, source code, and payload scripts required to automate USB attacks.
- **DIP switch (x4)**: Boot mode selector, providing 16 possible options. Different payloads may be configured for each boot mode.
- **Tactile push button (x2)**: Provides user interaction that allow executing payloads on-demand based on the selected boot mode.
- **RGB LED light**: The light allows our device to communicate with the user. These may indicate, for example, when a script is being executed and if the execution was a success or a failure.
- **330 resistor (x3)**: Limits the current flowing through the LEDs to prevent them from burning out.
- **Raspberry Pi Zero USB stem**: Provides the USB interface connector for host devices. Alternatively, a micro-USB cable may be used instead.

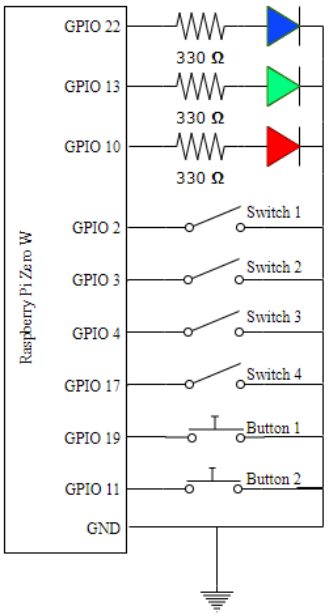

Figure 1
Device Schematic

Software Components

The software components required are freely available and/or open-source. The required scripts can be categorized within three main purposes: device setup, payload launcher service, and payload tools.

- **Operating System - Raspbian Stretch Lite:** Minimal version of Raspbian, a computer operating system based on Debian Stretch. Raspbian is Raspberry Pi Foundation's official operating system.
- **Device Setup:** Consists of a single 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. Figure 2 demonstrates how the modules can be enabled.

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

Figure 2
Bash Commands to Enable USB Gadget Configuration

- **Payload Launcher Service**: Consists of a single 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 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

We will be implementing attack vectors 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. We will also make use of the LibComposite kernel module, which allows greater control over the configuration of gadgets.

The USB device descriptors must be set before establishing the functionality required for the implementation of any attack vector. Figure 3 demonstrates a basic template for configuring the USB device descriptors. Even though we are using generic identifiers, we can observe that spoofing valid vendors and products is as simple as specifying the corresponding IDs. These IDs can be easily obtained through publicly available lists [20].

```
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
```

Figure 3
Template for Configuration USB Device Descriptors

HID Attack Vector

Devices masquerading as HID can easily emulate user input. These attacks are highly versatile, as they allow for the replication of mostly any action that a user can perform through a HID device. Most attacks in this category rely on batch scripts and typically aim to emulate HID keyboards due to the range of input options they provide. Figure 4 demonstrates how we can implement the functions required for the emulation of a HID keyboard.

```
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/
```

Figure 4
Configuration of Functions for Emulation of USB HID Keyboard

In Windows operating systems, HID attacks are typically initiated through the "Run" prompt, which facilitates access to a variety of applications. Some of the most commonly used tools for performing these attacks are the command prompt and PowerShell, which provide powerful scripting interfaces that are available on most Windows computers.

Figure 5 demonstrates a simple payload script for performing a HID attack that takes advantage of the Ducky Script interpreter tool to simplify the

syntax. The script starts off by setting up our device as a HID device. It continues to open the command prompt through the Run window, followed by the Notepad executable, in which it proceeds to type in some text. The script finalizes by flashing the green LED to indicate success. Figure 6 shows the results on the target computer.

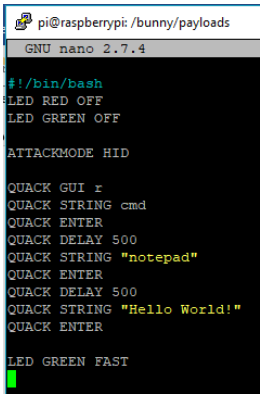

Figure 5
Example of a Payload Script for Execution of a HID Attack

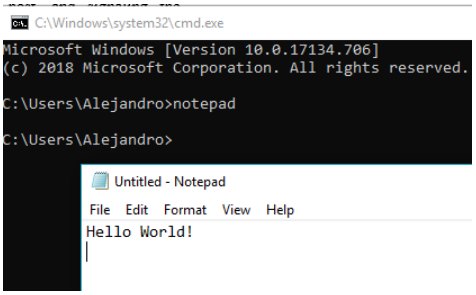

Figure 6
Result of HID Attack on the Victim Computer

Storage Device Attack Vector

The storage attack vector allows simulating a USB storage device. This mode is typically used in conjunction with other attacks to aid in the infiltration and exfiltration of data or files. In the setup of our device, a default 128MB FAT disk image is created that can later be mounted into the target computer when the storage attack mode is activated. Figure 7 demonstrates how we can implement the functions for emulating a USB storage device by using an existing disk image.

Figure 8 demonstrates a basic payload script that combines the HID and storage attack vectors. The script introduces an executable script into the host that is used to produce and exfiltrate information. The script starts off by synchronizing a PowerShell payload script into a disk image, which is then mounted into the host by initiating the storage device attack mode. Afterwards, a HID attack is performed that uses the host's PowerShell command line interface to execute the mounted payload script. The script generates a text file containing the folder structure of the current user's documents folder which is then written directly into our device's mounted disk for exfiltration. Figure 9 demonstrates the resulting text document contained within the mounted disk.

```
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/
```

Figure 7
Configuration of Functions for Emulating a USB Storage Device

```
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
```

Figure 8
Payload Script Combining HID and Storage Attack Vectors

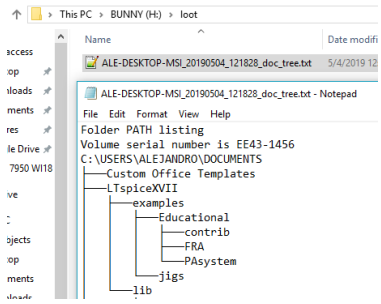

Figure 9
Result of Combined HID and Storage Device Attacks on the Victim Computer

Ethernet Adapter Attack Vector

The functionality of USB Ethernet adapters can easily be emulated and used as another attack vector. When the attack mode is enabled, a

subnetwork is created in which both the device and the host can communicate through unique IP addresses. A USB network interface and a Dynamic Host Control Protocol (DHCP) server are enabled in the device. At the same time, the device also acts as a network interface on the host, which automatically receives an IP from the devices DHCP server. Figure 10 demonstrates how we can implement this behavior.

```
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
```

Figure 10
Configuration of Functions for Emulating a USB Ethernet Adapter

This attack vector provides ample flexibility for implementation of attacks. By itself, it is typically used to perform reconnaissance on a target. Interaction performed solely through network communications is usually limited, especially if proper controls have been implemented to prevent network attacks. But when combined with other attack vectors such as HID, the results can be devastating. A high level of control can be achieved on the target and its interactions with the device, yielding almost limitless possibilities. Reverse shells, for example, can be highly dangerous attack method that can be easily implemented through these attack vectors, giving the attacker complete access to the target's command line interface (CLI) through a TCP connection.

COUNTERMEASURES

Currently, there is no bulletproof solution for properly defending against possible USB attacks, as most solutions can be bypassed. But implementing controls will still aid in minimizing risk. As with most security solutions, establishing multiple layers of security by implementing multiple controls will provide the best results. The following are possible solutions that can be implemented in conjunction:

- **Prevent physical access**: The most effective countermeasures against USB attacks is to simply to prevent physical access to the USB interfaces. If possible, unused USB ports should be removed and possible access to remaining ports should be controlled.
- **Software controls**: Multiple approaches may be implemented through software to minimize risk of USB attacks. These approaches will aid in reducing risk of attacks but require leveraging security against accessibility and convenience. Some methods include disabling USB ports, blacklisting/whitelisting device types and vendors, limiting the number of devices of the same type that can be connected at a time, and USB device authorization. Other methods can rely on the detection and prevention of possible attacks. DuckHunter [21], for example, detects HID attacks by identifying suspicious typing speeds and proceeds to stop their execution.
- **Ban unauthorized devices**: Policies can be established for preventing users from bringing or using unauthorized devices. This approach can aid in the visual identification of suspicious devices.
- **Employee awareness**: Properly educating employees to be wary of possible USB attacks can help minimize their incidence. This approach can also aid in the identification of possible malicious devices and attackers.
- **Enforce screen lock policies**: Many USB attacks require access to a logged in user and can be thwarted by simply enforcing screen locks when a computer is not in use. This can be implemented through a combination of network policies and user education.

CONCLUSION

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

[1] Hak5. (2019). *Bash Bunny* [Online]. Available: https://shop.hak5.org/products/bash-bunny.

[2] Honeywell International Inc. (2018, October). *Industrial USB Threat Report* [Online]. Available: https://www.automation.com/pdf_articles/honeywellps/Honeywell-USB-Threat-Report.pdf.

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

[4] C. Cimpanu. (2018, Mar. 18). *Here's a List of 29 Different Types of USB Attacks* [Online]. Available: https://www.bleepingcomputer.com/news/security/heres-a-list-of-29-different-types-of-usb-attacks.

[5] Hak5. (2019). *USB Rubber Ducky* [Online]. Available: https://shop.hak5.org/products/usb-rubber-ducky-deluxe.

[6] Irongeek. (2010, Mar.). *Programmable HID USB Keystroke Dongle: Using the Teensy as a Pen Testing Device* [Online]. Available: http://www.irongeek.com/i.php?page=security/programmable-hid-usb-keystroke-dongle.

[7] S. Kamkar. (2014, Dec.). *USBdriveby* [Online]. Available: http://samy.pl/usbdriveby.

[8] D. Spill, M. Ossmann, and K. Busse. (n. d.). "TURNIPSCHOOL," in *nsaplayset.org* [Online]. Available: http://www.nsa playset.org/turnipschool.

[9] I. Ilascu. (2018, Aug.). USBHarpoon is a BadUSB Attack with a Twist", *bleepingcomputer.com* [Online]. Available: https://www.bleepingcomputer.com/news/security/usbharpoon-is-a-badusb-attack-with-a-twist.

[10] Arduino. (2019). *Arduino* [Online]. Available: https://www.arduino.cc.

[11] PJRC. (n. d.). *Teensy* [Online]. Available: https://www.pjrc.com/teensy.

[12] Raspberry Pi Foundation. (n. d.). *Raspberry Pi* [Online]. Available: https://www.raspberrypi.org.

[13] Security Research Labs. (2018). "BadUSB Exposure," in *SRLabs Open Source Projects* [Online]. Available: https://open source.srlabs.de/projects/badusb.

[14] Common Vulnerabilities and Exposures. (2019). *CVE-2010-2568* [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568.

[15] USB Kill. (2019). *USB Killer V3* [Online]. Available: https://usbkill.com/products/usb-killer-v3.

[16] GitHub. (2019). *Hak5 - Payload Library for the Bash Bunny* [Online]. Available: https://github.com/hak5/bash bunny-payloads.

[17] GitHub. (2019). *Hak5 - USB Rubber Ducky Payloads* [Online]. Available: https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads.

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

[19] Raspberry Pi Foundation. (n. d.). *Raspberry Pi Zero W* [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero-w.

[20] S. Gowdy. (2019, May 9). *List of USB ID's* [Online]. Available: http://www.linux-usb.org/ usb.ids.

[21] P. Sosa. (2019). "DuckHunter," in *GitHub.com* [Online]. Available: https://github.com/pmsosa/duckhunt.