Securely Transferring Data from BadUSB Devices

Marian TICU
University Politehnica of Bucharest, Romania
marianticu89@gmail.com

Abstract: The constant growth and improvement of BadUSB devices pose an increasing threat from a cybersecurity perspective. Nowadays, regular USB devices could turn into BadUSB devices by reprogramming the firmware. They achieve new features and can simulate other peripheral devices like a keyboard or an external network adapter which could be used for malware purposes: keystroke injection, malware delivery, data exfiltration, network hijack or electrical damage. The manuscript addresses the topic of Bad USB devices and proposes a mitigation solution based on a Raspberry Pi system with a custom-made kernel which limits the attack surface by removing certain kernel modules.

Keywords: BadUSB, Raspberry Pi, USB data transfer, USB HID attack, USB security

INTRODUCTION

First developed in 1996 by USB-Implementers Forum (USB-IF) in order to establish a universal communication protocol between hosts and devices, over the time, the USB protocol has gained a widespread popularity through its speed performances and small form factor interface. In the nowadays, the USB interface it is being used to connect mostly all known peripheral devices (Ex: printers, storage devices, audio-video devices, I/O devices, printers, external cards, etc.) to a large set of hosts (Ex: servers, PCs, tablets, smartphones, etc.). In terms of performance, it offers a very high bandwidth that is currently reaching speeds of up to 10 Gbit/s USB 3.1, 20 Gbit/s USB 3.2 *[USB-IF, 2017]* or 40 Gbit/s USB 4 *[USB-IF, 2020]*.

Apart from the obvious advantages, the spread use of the USB protocol has created new opportunities for hackers to bypass traditional security policies. In 2014, Nohl, Krißler and Lell demonstrated at BlackHat in Las Vegas that USB devices can be reprogrammed to emulate other types of devices like keyboards *[Nohl, Krißler, & Lell, 2014]*. Basically, an attacker could turn an existing USB hardware into BadUSB, by rewriting its microprocessor firmware with an infected version. The attack makes use of the microprocessor firmware update function and it is possible because current operating systems don't verify the firmware on the USB devices. A regular USB memory stick device can be reprogrammed to act like another device, like a keyboard. Afterwards, the operating system recognizes the device as a keyboard and acts accordingly with the commands received just as if a real user typed them.

Since then similar BadUSB devices have been developed with more added features capable of launching powerful cyber-attacks, such as man-in-the-middle attacks. These devices can be either built with ease using resources publicly available on the Internet or can be bought from different manufactures. The ease of obtaining such a device combined with the very low cost of production or purchase creates new opportunities for hackers to infiltrate inside computer networks.

One of the well-known manufacturers is Hak5 LLC *[Hak5 LLC, 2020]*, which can offer special crafted pentest tools that can be used for white hacking activities, meaning only authorized auditing sessions. One of their products, the Rubber Ducky is a regular USB memory stick looking device that can emulate a keyboard which can be used for keystroke injection to a victim guest. The commands which can be delivered by the device can be configured through a proprietary scripting language called Ducky Script. In the hands of a hacker, once plugged into a computer, this affordable pentest tool can have devastating effects which include command execution, malware delivery or data exfiltration. Similar products from the same manufacturer, with more evolved features, are BushBunny, KeyCroc and LanTurtle. The BashBunny is a Linux powered device with remote shell access, capable of mimicking serial, storage, keyboards and network cards. The KeyCroc is a keylogger and when specific keywords are typed it is capable to run configured payloads. It can also be used for remote shell. The LanTurtle looks like any other usual external network adapters except the fact that it can provide remote access and can run man-in-the-middle attacks *[Hak5 LLC, 2020]*.

Apart from the commercial solutions, there are also multiple open source resources that could be used to craft a BadUSB device with ease. One of them is called TurnipSchool *[Dominic Spill, Michael Ossmann, Jared Boone, 2015]*. The specificity of this device consists in the fact that it is integrated in a USB cable and can be controlled by radio.

The spread of BadUSB devices, the ease of building one from scratch, like Spyduino *[Karystinos, Andreatos, & Douligeris, 2019]*, or buying commercial products, like Rubber Ducky *[Hak5 LLC, 2020]* or Malduino *[MalDuino, 2020]*, introduces a cyber security risk that must be addressed and managed accordingly. This kind of threat creates multiple attack vectors opportunities that can bypass traditional network security measures.

The visions of IT&C security companies against BadUSB attacks focus around the ideas of providing a software solution that whitelists trusted USB devices and blocking all the others *[Endpoint Protector, 2020]* or providing trusted firmware signed USB devices that can be accesed through dedicated software *[Iron Protector, 2020]*.

Without a BYOD *[B. Alotaibi and H. Almagwashi, 2018]* restricting policy in place, users could bring already infected personal devices and connect them to any asset in the internal network, which could lead to compromising an organization's IT&C infrastructure by bypassing all security layers of protection. Most of the times, data is stored on USB external storage devices with an unknown origin because they are delivered by third parties the organization works with. Organizations need to update and strengthen their security policies to withstand USB cyber threats. On each system, only required peripheral devices should be allowed functioning based on a whitelist *[Nohl, Krißler, & Lell, 2014]* and all other available USB ports should be disabled or blocked. At the organization level, several points of transfer must be established with designated systems, usually these are dedicated workstations for transferring data inside and outside of the network.

This paper presents a cost-efficiently approach of building a data transfer device that could be used by organizations or individuals to protect their networks against USB commonly known cyber-attacks. This system should be placed in frontline, the first system that an untrusted USB device should be first plugged in to.

METHODS

The USB protocol groups the USB peripheral devices according to their functionalities and organize them in a hierarchy of classes, subclasses and protocols *[Axelson, 2015]*. For example, a keyboard or a mouse connected through USB will be associated to HID or Human Interface Device class while a memory stick will be linked with the Mass Storage class. This information is used by the host operating system to assign the required driver for proper usage of the peripheral device capabilities. In a Linux OS, a device driver can be directly integrated into kernel or loaded as a module. By removing specific device drivers, some device functionalities would be inhibited.

Most of the rogue USB devices special crafted to be used as cyber-attack vectors, like BashBunny produced by Hak5, are successful because they are able to disguise themselves as a keyboard or external network card, and the OS to which they are connected loads the kernel modules responsible for HID or Communications and CDC Control, ensuring them full operability.

This paper presents a method of building a performant, accessible, and cost-efficient data transfer device immune to BadUSB cyber-attacks by recompiling OS kernel without the modules that add support for USB features which might pose a security threat by keeping only the module responsible for Mass Storage devices support.

The data transfer system consists of a small form factor computer, the Raspberry Pi Model 3B+. The keyboard and mouse will become unfunctional because HID support is going to be removed from kernel. A Raspberry compatible touchscreen will be used as an input method because it uses a separate kernel module and it still keeps working after removing the HID support. If a touchscreen is not available, the system can be controlled through SSH or remote desktop protocols via network.

The Linux kernel has a modular structure, meaning that it could adapt to various hardware configurations by integrating only the required modules. The integration can be done statically, which means that modules are built into the kernel and can't be removed afterwards, or dynamically, which gives the possibility to add or remove modules from the kernel on-the-fly while it is running.

The kernel module responsible for HID support is called *usbhid [Hallinan, 2010]*. This can be removed in two possible ways:

a) If the *usbhid* module is dynamically built as loadable module it can be removed using any of the Linux utils: *modprobe* or *udev*;

b) If the *usbhid* module is statically built into the kernel, then the kernel needs to be recompiled from sources. While the comfortable way is to unload the *usbhid* module whenever it is needed to, the safest solution is to build a kernel without it.

There are two ways of building a kernel from sources. It can be done directly on the system that we intend to rebuild the kernel for or it can be cross-compiled on another machine.

The entire process for cross-compiling Raspberry Pi kernel *[Foundation, 2020]* requires the following steps:

a) Download the kernel source code from official Raspberry Pi repository on GitHub.

```
# git clone --depth=1
https://github.com/raspberrypi/linux
```

b) Download the tools required for the building processing.

```
# git clone https://github.com/raspberrypi/
tools ~/tools
```

c) Update PATH (for x64 systems)

```
# echo PATH=\$PATH:~/tools/arm-
bcm2708/gcc-linaro-armlinux-gnueabihf-
raspbian-x64/bin >> ~/.bashrc
```

d) Install or update the required dependencies

```
# sudo apt-get install git bison flex
libssl-dev
```

e) Generate default configuration file (.config).

```
# KERNEL=kernel7
```

# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihfbcm2709_defconfig

f) Customize configuration through:

- Command line interface

  # make config

- Graphical User Interface tools like *menuconfig* or *xconfig*.

  # make ARCH=arm CROSS_ COMPILE=arm-linux-gnueabihf-menuconfig

  # make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-xconfig

Even though both *menuconfig* and *xconfig* are similar, the *xconfig* (***Figure 1***) is more user-friendly, better organized and it displays additional information about kernel modules. Through *xconfig*, the user can configure the modules that can be included in the kernel as static modules, the modules that can be compiled as loadable modules or the modules that can be excluded.

By default, the *usbhid* kernel module is built-in into the Raspberry Pi's kernel. In ***Figure 1***, the USB HID Support was disabled. This module can be found on the following path of the kernel structure: *Device Drivers → HID Support → USB HID Support → USB HID transport layer*.

The Raspberry Pi's touchscreen kernel module (***Figure 2***) must be installed because the keyboard and mouse support will be removed and the touchscreen will get to be the only method of user interaction available.

Beware of the unnecessary changes of the configuration because even the smallest ones can lead to a non-functional kernel.

g) Save configuration and start compiling the kernel

# make -j 12 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihfzImage modules dtbs

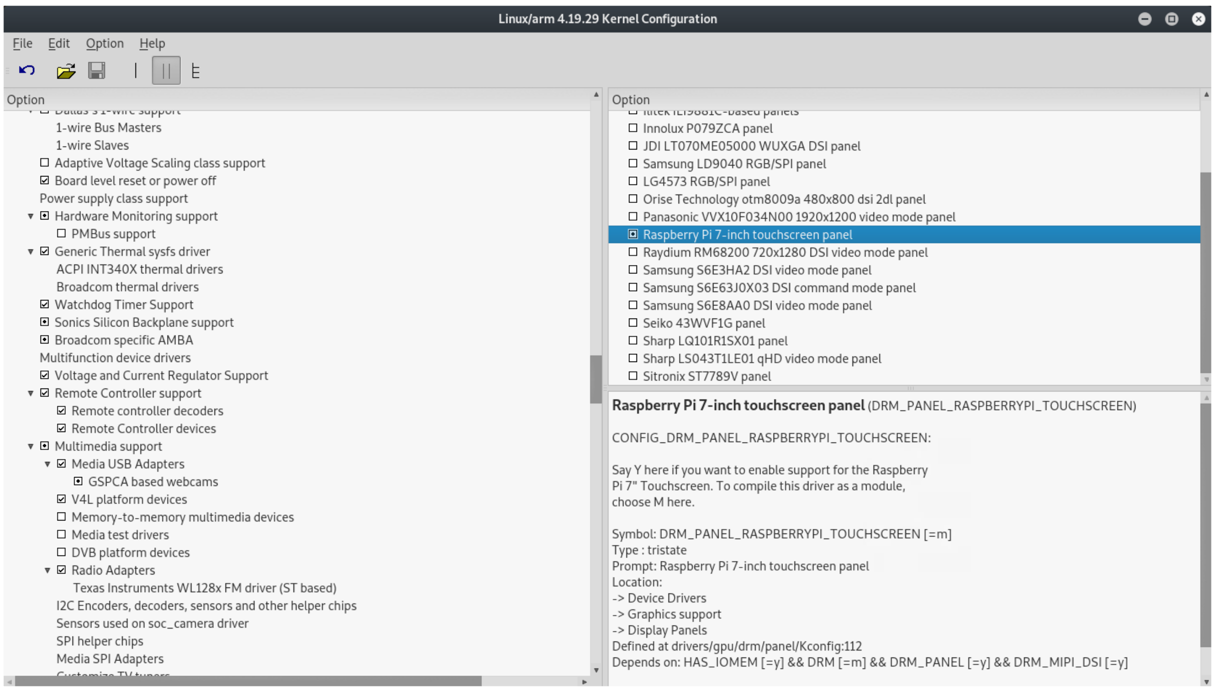
The procedure to install the newly cross-compiled kernel *[Foundation, 2020]* on Raspberry Pi 3 model B+ follows the next steps:

h) Remove the microSD card from Raspberry, connect it to the computer and mount the existing partitions. It should have two partitions by default: one for the boot order configured as FAT32 and one as root file system configured as EXT4.

# mount /dev/sdb1 /mnt/fat32

# mount /dev/sdb2 /mnt/ext4

i) Install modules



```
      1-wire Bus Masters
      1-wire Slaves
    ☐ Adaptive Voltage Scaling class support
    ☑ Board level reset or power off
    Power supply class support
  ▾ ⊡ Hardware Monitoring support
      ☐ PMBus support
  ▾ ☑ Generic Thermal sysfs driver
      ACPI INT340X thermal drivers
      Broadcom thermal drivers
    ☑ Watchdog Timer Support
    ⊡ Sonics Silicon Backplane support
    ⊡ Broadcom specific AMBA
    Multifunction device drivers
    ☑ Voltage and Current Regulator Support
  ▾ ☑ Remote Controller support
      ☑ Remote controller decoders
      ☑ Remote Controller devices
  ▾ ⊡ Multimedia support
    ▾ ☑ Media USB Adapters
        ⊡ GSPCA based webcams
      ☑ V4L platform devices
      ☐ Memory-to-memory multimedia devices
      ☐ Media test drivers
      ☐ DVB platform devices
    ▾ ☑ Radio Adapters
        Texas Instruments WL128x FM driver (ST based)
      I2C Encoders, decoders, sensors and other helper chips
      Sensors used on soc_camera driver
      SPI helper chips
      Media SPI Adapters
```

```
☐ Innolux P079ZCA panel
☐ JDI LT070ME05000 WUXGA DSI panel
☐ Samsung LD9040 RGB/SPI panel
☐ LG4573 RGB/SPI panel
☐ Orise Technology otm8009a 480x800 dsi 2dl panel
☐ Panasonic VVX10F034N00 1920x1200 video mode panel
⊡ Raspberry Pi 7-inch touchscreen panel
☐ Raydium RM68200 720x1280 DSI video mode panel
☐ Samsung S6E3HA2 DSI video mode panel
☐ Samsung S6E63J0X03 DSI command mode panel
☐ Samsung S6E8AA0 DSI video mode panel
☐ Seiko 43WVF1G panel
☐ Sharp LQ101R1SX01 panel
☐ Sharp LS043T1LE01 qHD video mode panel
☐ Sitronix ST7789V panel
```

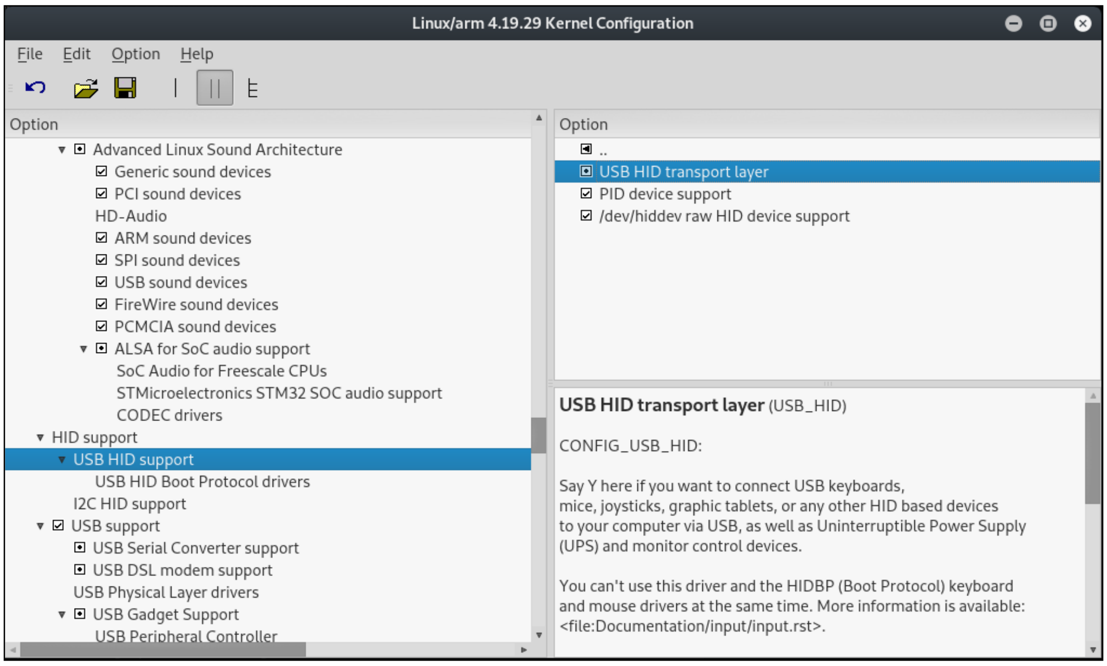
```
Raspberry Pi 7-inch touchscreen panel (DRM_PANEL_RASPBERRYPI_TOUCHSCREEN)

CONFIG_DRM_PANEL_RASPBERRYPI_TOUCHSCREEN:

Say Y here if you want to enable support for the Raspberry
Pi 7" Touchscreen. To compile this driver as a module,
choose M here.

Symbol: DRM_PANEL_RASPBERRYPI_TOUCHSCREEN [=m]
Type : tristate
Prompt: Raspberry Pi 7-inch touchscreen panel
Location:
-> Device Drivers
-> Graphics support
-> Display Panels
Defined at drivers/gpu/drm/panel/Kconfig:112
Depends on: HAS_IOMEM [=y] && DRM [=m] && DRM_PANEL [=y] && DRM_MIPI_DSI [=y]
```

Fig. 1: *Using xconfig to exclude kernel modules*



Linux/arm 4.19.29 Kernel Configuration

File Edit Option Help

```
Option
  ▾ ⊡ Advanced Linux Sound Architecture
      ☑ Generic sound devices
      ☑ PCI sound devices
      HD-Audio
      ☑ ARM sound devices
      ☑ SPI sound devices
      ☑ USB sound devices
      ☑ FireWire sound devices
      ☑ PCMCIA sound devices
    ▾ ⊡ ALSA for SoC audio support
        SoC Audio for Freescale CPUs
        STMicroelectronics STM32 SOC audio support
        CODEC drivers
▾ HID support
  ▾ USB HID support
      USB HID Boot Protocol drivers
    I2C HID support
▾ ☑ USB support
    ⊡ USB Serial Converter support
    ⊡ USB DSL modem support
    USB Physical Layer drivers
  ▾ ⊡ USB Gadget Support
      USB Peripheral Controller
```

```
Option
  ..
⊡ USB HID transport layer
☑ PID device support
☑ /dev/hiddev raw HID device support
```

USB HID transport layer (USB_HID)

CONFIG_USB_HID:

Say Y here if you want to connect USB keyboards,
mice, joysticks, graphic tablets, or any other HID based devices
to your computer via USB, as well as Uninterruptible Power Supply
(UPS) and monitor control devices.

You can't use this driver and the HIDBP (Boot Protocol) keyboard
and mouse drivers at the same time. More information is available:
<file:Documentation/input/input.rst>.

Fig. 2: Removing USB HID support

# env PATH=$PATH make ARCH=arm CROSS_COMPILE=arm-linuxgnueabihf- INSTALL_MOD_PATH=/mnt/ext4 modules_install

j) Back-up existing kernel

# KERNEL=kernel7

# cp /mnt/fat32/$KERNEL.img /mnt/fat32/$KERNEL-backup.img

k) Install new kernel and copy all required files

# cp ./arch/arm/boot/zImage /mnt/fat32/$KERNEL.img

# cp ./arch/arm/boot/dts/.dtb /mnt/fat32/*

# cp ./arch/arm/boot/dts/overlays/.dtb* /mnt/fat32/overlays/*

# cp ./arch/arm/boot/dts/overlays/ README /mnt/fat32/overlays/

l) Unmount partitions and reinsert microSD into Raspberry and boot up.

# umount /mnt/fat32

# umount /mnt/ext4

The device can be used for transferring data as follows:

a) The Raspberry Pi standalone:

– The operating system can be controlled via touchscreen;

– The user can attach up to four storage devices if it uses only the existing USB ports on the Raspberry Pi or it can expand to more than four using a USB hub;

– The user can copy data between devices.

b) The Raspberry Pi connected to a PC through Ethernet.

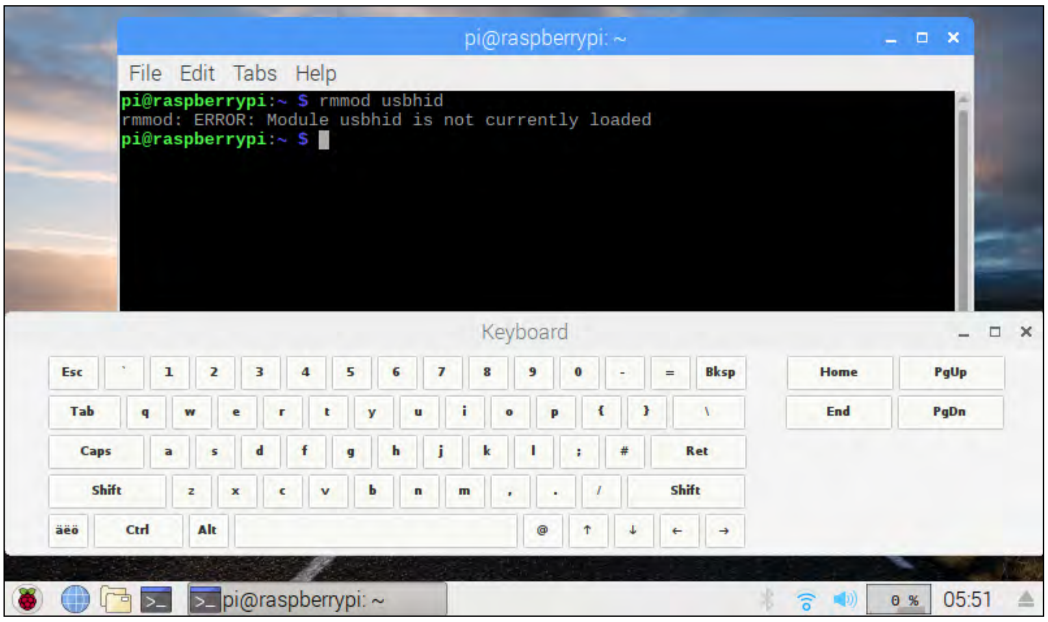
– For an improved level of security, it is advised that the computer stands as an isolated system;

– The user can control the Raspberry Pi interface from the computer using regular remote access protocols (VNC, SSH);

RESULTS

By means of the method highlighted in the current paper, a safe data transfer device was crafted by modifying the OS kernel of a small form factor computer, the Raspberry Pi. The HID support was removed directly from the kernel which makes the device a safe system for plugging suspicious USB devices from untrusted recipients in order to read or write data. This solution prevents BadUSB cyber-attacks that emulates human input devices like keyboards, joysticks or mouses.

Since the keyboard or mouse would not function anymore, the user has the possibility to interact with the data transfer device via touchscreen by using a virtual keyboard


pi@raspberrypi: ~
File Edit Tabs Help
Keyboard
Esc ` 1 2 3 4 5 6 7 8 9 0 - = Bksp
Tab q w e r t y u i o p [ ] \
Caps a s d f g h j k l ; # Ret
Shift z x c v b n m , . / Shift
äëö Ctrl Alt @ ↑ ↓ ← →
Home PgUp
End PgDn
pi@raspberrypi: ~
0 %
05:51

```
pi@raspberrypi:~ $ rmmod usbhid
rmmod: ERROR: Module usbhid is not currently loaded
pi@raspberrypi:~ $
```

Fig. 3: *Virtual keyboard*

(***Figure 3***) or over a TCP/IP connection through remote desktop or SSH protocols.

The same process could be extended to remove any unneeded kernel module that might pose a security risk, like the CDC module that adds support for external network USB adapters.

In terms of performance, the Raspberry Pi 3 model B+ is equipped with four USB 2.0 ports, theoretically sustain a data transfer at speeds up to 480 megabits per second.

The device was tested using Hak5 BashBunny configured to launch a HID attack that will run a reconnaissance script on the target host and save the output results on Bashbunny's storage space. BashBunny is a mini-computer with an USB memory stick form factor. It is running Linux operating system and it is powered by a quad-core ARM Cortex A7 processor, 512 MB DDR3 RAM memory and 8 GB storage. The BashBunny can emulate some well-known trusted USB devices (network card, serial communication, flash storage, keyboard). When it is used to simulate a keyboard, its actions can be configured with a custom payload written in Bunny Script. The device is very popular among cyber security penetration testing experts and has a wide range of payloads available.

The payload written in *BunnyScript* is:

```
# Linux Reconnaissance
LED SETUP
ATTACKMODE HID STORAGE
GET SWITCH_POSITION
LED ATTACK
Q ALT F2
Q DELAY 500
Q STRING gnome-terminal
Q DELAY 500
Q ENTER
Q STRING export
output=/media/\$USER/BashBunny/loot/
LinuxRecon.txt
Q DELAY 500
Q ENTER
Q STRING export
runscript=/media/\$USER/BashBunny/
payloads/$SWITCH_POSITION/linuxrecon.sh
Q DELAY 500
Q ENTER
Q STRING bash \$runscript \$output
Q DELAY 500
Q ENTER
LED FINISH
```

The content of *linuxrecon.sh* is:

```
#!/usr/bin/env bash
echo -e „Linux Reconnaissance \n\r” > $1
echo -e „-------------------- \n\r” >> $1
echo -e „Running processes \n\r” >> $1
ps -ax >> $1
echo -e „\n\r”
echo -e „-------------------- \n\r” >> $1
echo -e „Network config \n\r” >> $1
```

```
ifconfig -a >> $1
echo -e „\n\r”
echo -e „--------------------- \n\r” >> $1
echo -e „Network connections \n\r” >> $1
netstat >> $1
echo -e „\n\r”
echo -e „--------------------- \n\r” >> $1
echo -e „Users \n\r” >> $1
cat /etc/passwd >> $1
echo -e „\n\r”
echo -e „--------------------- \n\r” >> $1
echo -e „USB \n\r” >> $1
lsusb >> $1
echo -e „\n\r”
echo -e „--------------------- \n\r” >> $1
sync
```

The safe data transfer device prevented BashBunny’s HID attack.

CONCLUSIONS

This paper presents a low cost efficient and safe data transfer device that could be used by any organization or individual to empower the procedures and cyber security policies regarding the access to the data from untrusted USB storage devices. Along with a security policy of blocking USB ports on all computers on the network, the procedure for transferring data from outside to inside and from inside to outside using USB storage devices must consider defining single points for data transfer that use safe data transfer devices.

The design simplicity and low costs of development make it accessible to IT Security Staff that could be using it to raise the cybersecurity resilience of their organizations.

ACKNOWLEDGMENTS
The work has been funded by the Operational Programme Human Capital of the Ministry of European Funds through the Financial Agreement 51675/09.07.2019, SMIS code 125125.

REFERENCE LIST

Axelson, J. (2015). USB Complete Fifth Edition. Madison, United States: Lakeview Research, U.S. Retrieved 2020, from https://en.wikipedia.org/wiki/USB
B. Alotaibi and H. Almagwashi. (2018). A Review of BYOD Security Challenges, Solutions and Policy Best Practices. 1st International Conference on Computer Applications Information Security (ICCAIS). Riyadh.
Dominic Spill, Michael Ossmann, Jared Boone. (2015). NSA Playset: USB Tools. ShmooCon Proceedings. Retrieved 2020, from NSA Playset: http://www.nsaplayset.org/turnipschool
Endpoint Protector (2020). Retrieved from https://www.endpointprotector.com/
Foundation, R.P. (2020). Raspberry Pi kernel building. Retrieved 2020, from https://www.raspberrypi.org/documentation/linux/kernel/building.md
Hak5 LLC (2020). Hak5. Retrieved 2020, from https://hak5.org
Hallinan, C. (2010). Embedded Linux Primer: A Practical Real-World Approach. Prentice Hall.
Iron Protector (2020). Retrieved from http://www.ironprotector.com/
Karystinos, E., Andreatos, A., & Douligeris, C. (2019). Spyduino: Arduino as a HID Exploiting the BadUSB Vulnerability. 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), (pp. 279-283).
MalDuino (2020). Retrieved 2020, from https://malduino.com/
Nohl, K., Krißler, S., & Lell, J. (2014). BadUSB - On accessories that turn evil. Preluat de pe Security Research Labs: https://srlabs.de/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf
USB-IF (2017). Universal Serial Bus 3.2 Specification. Retrieved from https://www.usb.org/document-library/usb-32-specification-released-september-22-2017-and-ecns
USB-IF (2020). Universal Serial Bus 4 (USB4™) Specification. Retrieved from https://www.usb.org/document-library/usb4tm-specification