Heimdall: USB mass storage devices threat evaluation using an embedded system

Ivan Zlatanov
ivanff123@gmail.com
Author
National Trading and Banking High School
Sofia, Bulgaria

Yavor Papazov
yavor@esicenter.bg
Mentor
CyResLab
Sofia, Bulgaria

ABSTRACT

The Universal Serial Bus is a serial communication standard that is implemented in most of the computer-like devices in the world. Despite its acceptance, government and private entities have partially of fully banned all devices that enforce the USB communication interface in their facilities. These security measures are in result of the increased USB cyberattacks. This paper explores and assess detection methods, developed for such attacks. By creating low cost testing device we are able to perform threat evaluation on potentially malicious USB mass storage devices without the risk of compromising or destroying real computer systems.

KEYWORDS

cybersecurity, USB, mass storage devices, threat evaluation

1 INTRODUCTION

The Universal Serial Bus (USB) is one of the most widely used serial communication standards which is enforced in almost any modern computer-like device. The concept behind it is to provide asynchronous interface for multiple external devices connected to one host.

A large proportion of the mass storage devices (MSDs) on the market use USB interfaces in order to connect to hosts that support the Universal Serial Bus mass storage device class standard. Generally, MSDs are lightweight, small, with high capacity and able to transfer large quantities of data in a matter of seconds. Although some of them are external hard drives and so are neither lightweight or small, the most frequently used MSDs are USB flash drives (also known as thumb or pen drives), which hold capacity thresholds up to 2TB and can last up to approximately one hundred years. These attributes make USB thumb drives perfect for long or short-term data storage.

Regardless of the popularity of the Universal Serial Bus and its interface, the usage of mass storage devices that implement it is restricted or fully banned in multiple government agencies and enterprises because of the security risks they pose. USB mass storage devices can and have been efficiently used to distribute malicious payloads in cybercrimes all around the world. The existence of flash drive alike devices such as USB killers and Rubber Duckies demonstrates how untrained personnel can permanently destroy segments of computer systems without even realizing it. Because of the reasons listed above the Pentagon, IBM, NASA and many more entities have banned the usage of removable USB storage devices in their facilities.

USB cyberattacks such as Stuxnet [6], Flame [7] and BadUSB [1] proved that mass storage can be used to exploit system vulnerabilities and cause catastrophic failures, resulting in immense damages. In order to reach maximum effectiveness, such attacks ought to be distributed to the highest possible number of computers.

In this paper we propose Heimdall - a stack of methods for USB mass storage devices analysis that are able to detect potential threats. Our approach relies on an embedded system that performs multiple tests, particularly designed to detect malicious behavioural characteristics in a USB mass storage device and thus to render it as dangerous.

2 RELATED WORK

USB mass storage and peripheral devices security is one of the most threatening problems in cybersecurity. Despite that, it is not properly addressed. Previous researches in the field concentrate on creating defence methods, while actually lacking proper threat evaluation approach. This produces inefficient systems that are not able to identify certain types of USB cyberattacks attacks.

Cinch [2] is an example for a defence system. It uses virtualization to logically separate the process of using USB devices between two computers. One of them acts as an untrusted machine to which USB peripheral devices can be connected and one that works on different logical level and only implements the functions of the connected devices without directly interacting with them.

However, Cinch has many problems in protecting against hardware attacks because its operationality is limited to the software level.

3 BACKGROUND

3.1 The Universal Serial Bus

The Universal Serial Bus (USB) is the most popular standard for serial connections between computers and peripheral devices. More than six standards and 3 connector types have been developed and used since its deployment in 1996. Designed as unification and simplification standard for the connection between personal computers and peripheral devices, it is now an essential part of every modern computer-like device.

3.2 USB Attack Vectors

Cybercrimes in the recent years have been affecting healthcare, transport, defence, banking and more industries. Forbes estimates six billion dollars in damages done by cyberattacks by 2021.

USB attacks are all cyberattacks that are performed by or with the help of devices that implement the Universal Serial Bus interface. They can be targeted or random. Such attacks may be utilized for data exfiltration, establishing command and control over systems, malware spreading or equipment destruction.

The most popular USB cyberattacks are Stuxnet, which infected facilities of the Iranian uranium enrichment program and delayed it by approximately one year and Flame, which was developed and used for cyber espionage in the Middle East. These attacks demonstrated the real-world damages that can be done via computer-like devices that implement the USB interface.

3.2.1 USB Drop Attack. Hackers used to place malicious USB drives in strategical locations since Stuxnet. The aim of this practice is to trick a potential victim into using it and therefore infecting or damaging his or hers system. This distribution method is called USB drop attack and it is used by cybersecurity researchers and attackers around the world. The USBs are usually placed in public locations - parking lots, hallways or common rooms. This way the chances of success are higher.

3.2.2 Power Surge Attack. A USB like devices called USB killers can be used to produce electrical circuit shortage in all hosts that have USB ports. Such devices induce power surges by collecting electricity from their hosts' ports and charging it into high-voltage capacitors. The capacitors are then discharged into the hosts' ports. The process keeps repeating until the ports are damaged and cannot supply power anymore. The objective of this attack is to destroy hardware components.

Power surges through USB ports have been performed on computers, smartphones, game consoles, printers, monitors and even vehicles. Figure 1 is a blueprint of a generic USB killer.

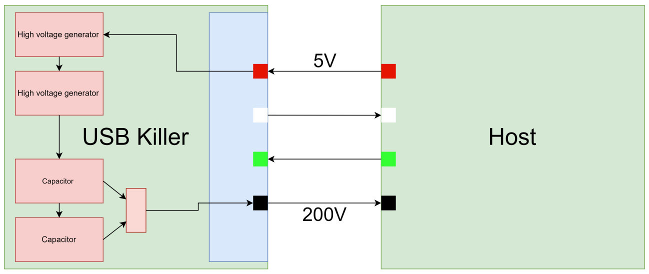

Figure 1: Simple USB killer blueprint.

3.2.3 Maliciously Alternated Firmware. The vendor information of a USB mass storage device is important for every host system because it is used to find and attach the appropriate driver. It is possible to alternate such information with leaked OEM tools like Patriot and thus to trick a host system into loading a wrong driver.

Firmware alternation can be used to transform a regular USB MSD into HID or BadUSBs.

3.2.4 Backdoored Live Boots of Linux Distributions. The inital ramdisk (initrd) is an inital temporary root file system that is mounted prior to a real one and provides capability of running programs from it. Subsequently another file system can be mounted from a different device. The one from the initial ramdisk is then moved to a specific directory and can be later unmounted.

The main idea behind the initrd it to allow startup to occur in two steps, where the kernel comes with as little builtin modules as possible. This, unfortunately, opens multiple attack vectors through malicious loadable kernel modules. Such modules can be used to provide persistent backdoors for rootkits and thus fully compromise computer systems.

4 METHODS

The sole purpose of this paper and its implementation is to design and asses USB mass storage devices threat evaluation methods in order to prevent potential attacks. We have not executed any of the attacks in the previous section on any other system than our test system.

4.1 Unplugging Simulation

Preventing a USB mass storage device connected to our test system to keep track of important system events as OS, clock time and driver changes is crucial to the testing process. In order to accomplish this we developed a method that relies on an external USB port extender. The role of this hardware component is to provide software power control over the USB port through GPIO pins and also to detect and prevent power surge attacks executed via USB killers or similar malicious devices. The extender uses relays to control the power of the port and replaceable fuses located on the data and ground lines of the port to prevent power surges. In case of overvoltage the cheap to substitute fuses will break and protect the delicate electrical components after them. This will indicate that the tested device is malicious. Figure 2 illustrates the external USB port extender.

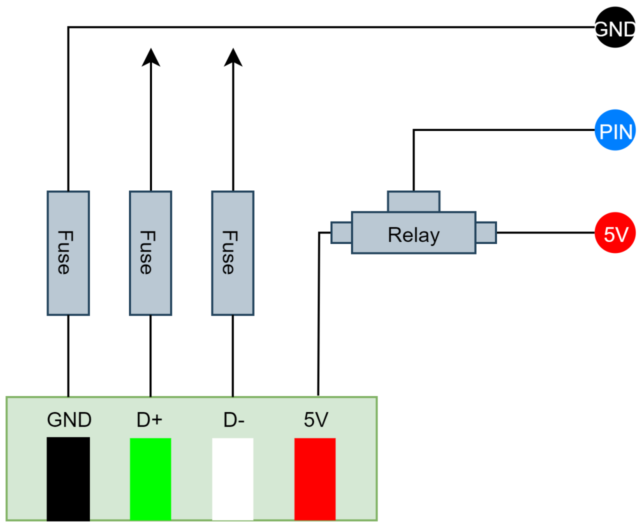

Figure 2: External port extender.

4.2 Device Type Validation

The most fundamental step of an adequate threat evaluation of a USB MSD is to determine if a device connected to a host is actually a mass storage device. We can do this by accessing and verifying the device's interface class which is stored inside its configuration. A configuration is the profile of a given device. Every configuration has one or more interfaces which correspond to the functionality provided by the device. USB drivers are typically written per interface, not per device. Devices can have more than one interface. Printers, for example, usually have at least two - one for printing and one for scanning. A device with Human Interface Device (HID) interface cannot be uses as a MSD and vice versa. A true USB mass storage device has only one interface, which is the mass storage interface.

The device validation test we developed finds the number of interfaces and their classes for a given device. It is therefore able to distinguish real USB mass storage device from a forged one.

4.3 Vendor Information Validation

As mentioned in the previous section, USB mass storage devices' firmware can be alternated in order to change their behaviour towards their host. The method we developed to detect such alternation accesses the vendor information on a device's firmware and saves it. We then simulate device unplugging and plugging, access the same data and compare it to the original one. By looping over this process multiple times we are able to determine if a MSD is changing its vendor information over different connections to a host.

4.4 Virus Scan

Generic virus scans are vital for detection of common malwares and rootkits in various files. We initiate a virus scan before advancing with further tests that include I/O operations because it is possible for an infection to occur during these actions.

4.5 File Poisoning Inspection

It is possible to detect an attempt for file poisoning by MSD's firmware if we generate, copy to the device, download from the device and compare a dump file. We randomly generate data and then save it as a file. The file is then copied to the connected USB mass storage device, downloaded from it and then compared to the original. Every difference between the files and their signatures is a solid proof that they have been alternated or at least tampered with.

Random generation of the data in the file is required in order to prevent falsification of the discussed test. For example, it is possible to design a malicious device that returns the same data (expected file content) after specific system calls or operations.

4.6 Live Boot Validation

We successfully developed a live boot validation test that operates in a networked or network isolated environment. Our method targets the live operating system Tails which focuses on privacy and anonymity.

In the first case the content of a live boot partition (LBP) is packed into a disk image file. The checksum (CS) of this file is then compared to the checksum of the current Tails version's. If internet connection is not present or authorized we use a list of the CSs of malicious initial ramdisks. We compare the content of this list to the checksum of the local initrd. Indicators for backdoored distribution are discrepancy between the local Tails image's checksum and the official or compatibility between blacklisted checksums and the local.

4.7 OS Emulation

Certain cyberattacks might target vulnerabilities and perform exploits on specific OSs. For example, an alternated mass storage device that targets Microsoft Windows might not work on Linux distributions and so the tests performed on them will be ineffective. For this reason emulating different operating systems is crucial for determining the true behaviour of a USB MSD.

In the interest of properly executing the suitable tests we use virtualization that emulates given OSs. Our method utilizes partial KVM [5] passthrough that provides direct link from the USB hub to the virtual machine (VM). This simply turns the VM into a userspace driver that ensures highest possible I/O performance for a device and a host. In this way we are able to collect behavioral data probes and then compare them to each other sufficiently.

5 IMPLEMENTATION

The practical implementation of our methods consists of two major parts - software and hardware. The software is a Python application that utilizes Libusb1, PyUdev and PyShark to access and monitor data from a device that is being tested. The hardware on which it runs is an enhanced Raspberry Pi 3 Model B+ [4] (RPI).

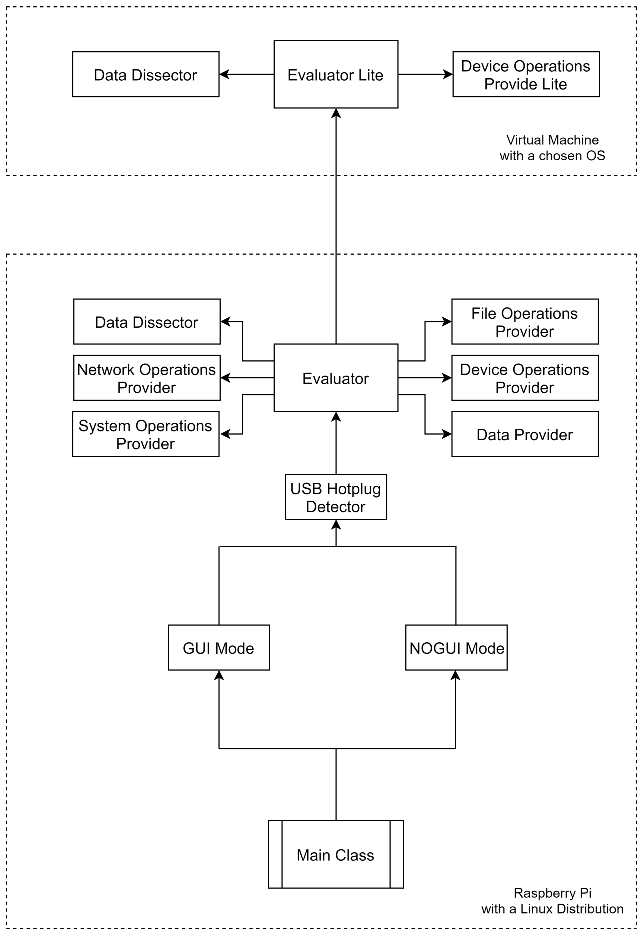
Data Dissector
Evaluator Lite
Device Operations Provide Lite
Virtual Machine with a chosen OS
Data Dissector
File Operations Provider
Network Operations Provider
Evaluator
Device Operations Provider
System Operations Provider
Data Provider
USB Hotplug Detector
GUI Mode
NOGUI Mode
Main Class
Raspberry Pi with a Linux Distribution

Figure 3: The software architecture of Haimdall.

5.1 Software

Our program, which architecture is illustrated on figure 3, starts with a one function main class that instantiates USBHotplugDetector class in GUI or NOGUI mode. The detector itself has stop() and start() functions which are responsible for its operationality. start() sets to true a variable called is_started and then calls the private function __begin_detecting(). Its purpose is to detect USB hotplug events by constantly looping and saving all block devices on the bus in a list of objects. The detector compares the list of every new iteration of the loop with the previous one. If the new list has more elements than the previous, then new devices have been plugged in. The difference of the list is the new devices tuple. Each of its elements is then handled according to their port number. Service devices that must use their kernel driver are on ports 0 and 2. Test devices use ports 1 and 3. They are forbidden from using their kernel drivers due to security risks. The driver for all test devices is libusb. After detection they are passed as parameters to the test_device function in the Evaluator class.

The Evaluator is responsible for performing all necessary tests and thus determining if a device is a threat to its host. validate_vendor_information() is the first performed test. It accesses and saves into variables the serial number (SN), binary-coded decimal number (BCDN), vendor id (VID) and product id (PID) of the tested device. A loop of unplugging simulations is then initiated. On each iteration the SN, BCDM, VID and PID are accessed and compared to the originals. Any difference is indication that a device is changing its vendor information during multiple connections to host.

The second performed test is validate_device_type. It verifies the true MSD character of the USB device. The validation process works by accessing, counting and inspecting the class and interface count of the device's configuration. The only acceptable result is one mass storage interface.

If we know that a device is indeed a mass storage device we can proceed and begin performing high level tests that involve its file system, if one is present. We first attach the device's kernel driver, mount it and then execute a generic virus scan using the Python module of an open-source malware detection engine called ClamAV [8]. After the scan the driver is detached.

If the previous test is successful the Evaluator performs __test_io() which examines the possibility of file poisoning by firmware. We attach the kernel driver again, mount the device and call generate_random_data_file() function from the DataProvider class that generates a 256 byte file named "dump.me" and saves it into its local directory. The file is copied to the mounted partition and then again moved to a its original directory with a new name - "received_dump.me".

The checksums and contents of both files are compared in order to determine if "received_dump.me" has been alternated.

Next is determining if the tested device contains live version of Tails. That is done by recursively searching the mounted partition for an inital ramdisk or an archived kernel file. If we do not find one the test is flagged as successful, but an extra information is presented in a log file. The __live-boot_validation() test operates in connected and network isolated environment. In the presence of internet connection the get_tails_checksum() is called from the NetwokrOperationsProvider class and the checksum of the current Tails version is downloaded. The contents of the live directory are packed into a disk image file and their checksum is compared with the genuine one. In case of network isolation the test relies on file called blacklisted.blk that contains checksums of known exploited initrds. Their checksums are compared to the local initrd's.

In order to ensure that the tested device is behaving properly under different operating systems we use virtual machine software called QEMU [3] to simulate Microsoft Windows. The VM has enabled KVM passthrough which allows direct connection to the USB Host Controller. We launch a lite version of our Evaluator which gathers data probes of the device's behaviour. This data is then written on the RPI's disk. The full Evaluator then takes a data probe of the behaviour of the device under Linux distributions and compares it with the Windows's. All OS specific data is ignored.

Every contributor and user of Heimdall can extend the scope of the implementation by writing own tests and saving them in the external_tests directory. All of them are executed right after the hardcoded.

If any of the discussed above tests fails it returns false boolean, as indication that the evaluated device is dangerous. How the user learns this depends on the mode in which the application works. In NOGUI mode it will be printed on their console. In GUI mode it will be outputed in the textbox bellow the "Logs:" label.

5.2 Hardware

As already mentioned in the beginning of this section, our application operates on an enhanced Raspberry Pi 3 Model B+ that runs Linux-based operating system. The most important part of our hardware, illustrated on figure 4, is the external USB port extender. It consists of a solid-state relay which is triggered by a GPIO pin and controls the power supply from the GPIO pin 5V to the port.

Two ampere fuses are placed on the ground and each data transfer line. They ensure that power higher than that will not penetrate and damage the internal components on the motherboard of our RPI.

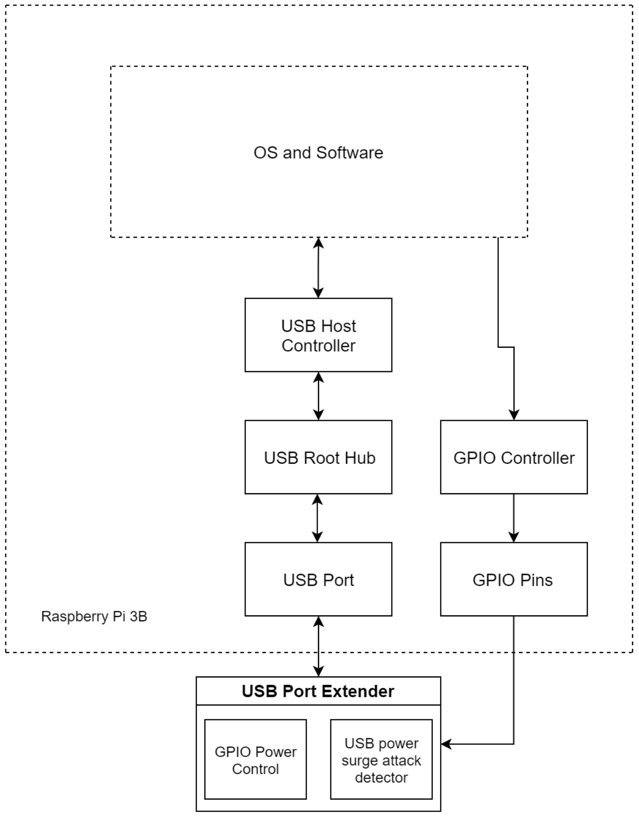

Figure 4: The hardware architecture of Haimdall.

6 RESULTS

We explored different cyberattacks executed by or with the help of devices that implement the USB interface and how they affect the government and private sector. We designed and implemented USB mass storage devices threat evaluation methods based on our findings. Furthermore, we developed and built hardware components that are able to detect and defend against USB killer induced power surges. Our software and hardware USB mass storage devices threat evaluation methods proved to be effective in a networked and network isolated systems.

7 FURTHER DEVELOPMENT

There is bright future for researches and projects that tackle cybersecurity problems related to the Universal Serial Bus because of its mass distribution and poor addressing. The methods and their implementation we developed are just a fraction of the possible outcomes of such projects.

We consider several further development directions for the project, namely:

- expanding the scope to not only mass storage devices;

- developing a custom USB kernel driver that performs tests on a connected device;
- expanding the project as a full framework.

ACKNOWLEDGMENTS

To Yavor Papazov and Konstantin Delchev, for the enormous amount of help with the choice of the research subject and the support during the development of Heimdall.

REFERENCES

[1] Laiali Almazaydeh, Jun Zhang, Peiqiao Wu, Yisheng Cheng, Khaled Elleithy, and Ruoqi Wei. Bad usb mitm: A network attack based on physical access and its practical security solutions. 2017.
[2] Sebastian Angel, Riad S. Wahby, Max Howald, Joshua B. Leners, Michael Spilo, Zhen Sun, Andrew J. Blumberg, and Michael Walfish. Defending against malicious peripherals with cinch.
[3] Fabrice Bellard. Qemu.
[4] The Raspberry Pi Foundation. Raspberry pi 3 model b+.
[5] kernelnewbies. Kvm.
[6] Ralph Langner. To kill a centrifuge. 2013. The paper explains what is Stuxnet and its way of opertaion.
[7] Laboratory of Cryptography and System Security. skywiper (a.k.a. flame a.k.a. flamer): A complex malware for targeted attacks. 2013.
[8] Cisco Systems. Clam antivirus (clamav).