System Protection Agent Against Unauthorized Activities via USB Devices

José Oliveira[1,2], Miguel Frade[3,2] and Pedro Pinto[1,4]
[1]*Instituto Politécnico de Viana do Castelo, Viana do Castelo, Portugal*
[2]*Computer Science and Communication Research Centre (CIIC), Polytechnic Institute of Leiria, Portugal*
[3]*DEI ESTG, Portugal*
[4]*INESC TEC, Porto, Portugal*

Keywords: USB, BadUSB, Microsoft Windows.

Abstract: Security attacks using USB interfaces and devices are becoming more advanced, which boost efforts to develop counter measures in order to protect systems and data. One of the most recent attacks using USB devices is the BadUSB attack, performed by spoofing the device's firmware and allowing the attackers to execute a set of malicious actions, e.g. an USB storage device could be mounted as USB keyboard in order to inject malicious scripts into the system. This paper proposes a protection agent against BadUSB attack developed for Windows operative systems. It allows a user to check the class of an USB device ready to be mounted, though enabling the detection of a potential attack if the expected functionality of the device does not match with its class type. The results show that the proposed protection agent is capable of detecting potential intrusions by blocking the installation of the device, scanning the device for something that identifies it, searching for a description locally and finally warning the user about the device meaning that all devices must be approved by the user when plugged in if the system protection agent is running.

1 INTRODUCTION

Nowadays there are many kinds of malicious hardware, such as back-doored network devices, back-doored firmware of devices (printers, hard drives, etc), keyloggers and many others. For the purpose of this work we focused only on Universal Serial Bus (USB) devices. USB devices are of special interest due to its high availability and also often require less user interaction to install, or no interaction at all. These characteristics enable USB devices to offer a large attack surface on computer systems (Jodeit and Johns, 2010).

In (Crenshaw, 2011) four categories are used to classify malicious USB devices, namely:

1. USB Mass Storage devices containing malware (malicious software);
2. U3 smart drives with malicious auto-run payloads;
3. Hardware key loggers
4. Programmable Human Interface Device (HID) USB Keyboard Dongle devices.

However, new types of devices have appeared since 2011 and thus, we propose the following updated set of categories:

1. USB Mass Storage devices containing malware;
2. U3 smart drives with malicious payloads;
3. USB devices in the Middle (USBiM);
4. USB with Programmable HID;
5. Denial of Service USB device;

The updated set of categories are based on those proposed in (Crenshaw, 2011) where the first two categories are the same, the third and fourth categories are renamed (to make them more general) and a fifth category is included.

Devices that fall into the first category are the standard way to present external storage to an host Operating System (OS). USB storage devices can be used as a way to inject malicious code to a computer (Terdiman, 2012; Walters, 2012) and, in case these devices are shared between multiple computers, the malicious code could be widely disseminated. In context of Microsoft Windows OS, and as stated in (Sood and Enbody, 2014), one of the most commonly exploited

Oliveira, J., Frade, M. and Pinto, P.
System Protection Agent Against Unauthorized Activities via USB Devices.
DOI: 10.5220/0006708502370243
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 237-243
ISBN: 978-989-758-296-7

vulnerability in this type of devices is the Windows OS autorun, which allows to execute automatically any code or software in a Windows OS.

The second category, ”U3 smart drives with malicious payloads”, groups the devices that have a special partition that is seen as a CD-ROM device when the device is enumerated by the host. Like the devices in the first category, these USB devices are also used as a means to deliver malware. The device is not malicious by itself, but it can contain files that are malicious in nature. However, in a mass storage USB device the infection might be accidental, but on a U3 smart drive the infection is generally intentional (Crenshaw, 2011).

The third category was renamed to ”USB devices in the Middle (USBiM)” to be able to group all devices that are inserted between a non malicious USB device and the host computer. The most common devices that fit this category are keyloggers; some of these are sold as forensic tools (TechGuru, 2017), however, other devices fit this category with malicious purposes, such as printer loggers, or hardware USB sniffers [1].

The fourth category, ”USB with Programmable HID”, represents a relatively new kind of threat where malicious code is embedded in device’s firmware to request USB interfaces that usually are not mass storage. This feature provides unacknowledged and malicious functionality that lies outside the apparent purpose of the device (Tian et al., 2015). The advantage of these type of devices is that with a USB HID it does not matter if autorun is disabled or not. By default, most OS seem to automatically install and configure USB HIDs as soon as they are inserted, regardless of the privilege level of the current user. Author in (Crenshaw, 2011) released a blog entry describing a programmable USB device that was capable of emulating a keyboard and ”typing” out commands specified in a script stored on the device. This technique allowed commands to be executed automatically by emulating a known keyboard type and vendor. Authors in (Cannon, 2010; Veres-Szentkiralyi, 2012; Benchoff, 2013) developed this concept to be able to allow perform data exfiltration through the USB-HID protocol. In 2016 (Kamkar, 2016) created another type of programmable HID called PoisonTap. This device uses a Raspeberry Pi Zero to emulate a fast gigabit network card that is able to exfiltrate data, even from locked computers.

Finally, the fifth category, ”Denial of Service USB devices”, is proposed to group multiple devices that have the malicious purpose to disrupt a service. One example is the USB killer (USBKill.com, 2017) that use the USB power lines to charge its capacitors, and then discharge 200V DC over the data lines of the host device, burning any circuit board that does not provide electrical surge protection [2].

The current paper proposes a system protection agent for Windows OS that focus into devices of the fourth category by identifying the device functionality and also, by scanning processes running on background originated from external devices. Devices from categories one and two can also be addressed by aiding the user to detect false USB devices, spoofed firmware or other threats to the system that might occur from USB devices. Our proposed solution intersects the operating system requests to install a device driver and hands over to the user the option to white list or blacklist the device based on the identified functionalities by the HID.

This paper is organized as follows. Section 2 presents the actual techniques used for protection against bad USB devices. Section 3 details the proposed system protection agent for Windows Operative System based on white listing of known good USB devices technique. Section 4 presents the validation procedures of the current proposal. Finally, section 5 presents the conclusions and future work.

2 BACKGROUND

Awareness of USB attacks is becoming more noticeable, since these devices can be used effectively to deploy malicious code in computers and networks where they connected. USB drives are a known security threat and they were already blamed for the installation of Conficker, a worm type of malware, on the Manchester City Council computers in the year 2009 (Andreasson, 2011) (Greene, 2010). There are many solutions to defend computer systems from malicious USB devices, but none of them solves the problem effectively without affecting the performance of the system or matching the end user requirements. We can define five general categories of defense approaches:

- malicious software detection;
- disable autorun;
- behavior detection;
- physical block of USB ports;
- device installation restrictions;

Malicious software detection and disabling autorun are the most common defense approaches

[1]Project to build a hardware USB sniffer https://github.com/dominicgs/USBProxy

[2]Video of USB Killer tests on different devices https://youtu.be/faKX_P1Be50

against malicious USB devices. These strategies work well for USB Mass Storage devices and U3 smart drives containing malicious payloads. According to (Crenshaw, 2011) these storage based classes of USB devices consist of two attack vectors. The autorun capabilities of different types of removable media make them different than malware that spreads over a network, or via a user deliberately choosing to run a binary, but autorun is already disabled by default in many modern OS. However, malware detection does not work against USB devices that fall into the last three categories of malicious devices.

Although some security software firms such as Symantec (Ulanoff, 2014) have recommendations about system protection against BadUSB attack, they do not provide an effective solution for these type of attacks.

A defensive mechanism against BadUSB attack would allow the user to simply disable USB inputs, notify user about a new plugged device, see what current processes are being originated and warn the user about it. Code signing for firmware updates could also provide protection but only if the signed firmware is modified. In this case, the device can not authenticate the adulterated firmware and therefore will not operate. On businesses, the threat is greater because a thumb drive can travel lots of computers per month or even the same day. Administrators can block USB ports or they can install software such as Symantec Endpoint Protection (Symantec, 2017) which has a device control module to prevent USB devices from mounting on the operating system (Ulanoff, 2014). There are other platforms that can prevent BadUSB attacks, such as Safend Protector, Sophos Endpoint Security and many others. As seen in (Cheng, 2014), another solution for BadUSB attacks is to adopt the cloud which can be used as a storage method to share information through the internet between company members and possible partners. In GoodUSB (Tian et al., 2015), the piece of code was capable of blocking the device and keep it waiting for user authentication to proceed with the natural functions of the device. It was a step further considering that it had its own USB Honeypot mechanism focused on BadUSB attack profiling.

Current paper proposes a protection agent against BadUSB attacks. This agent identifies the class of the USB device connected to the computer with Windows Operating System, and allows the user to block it based on the expected device type. During the driver installation, Windows Operating System obtains information from the device such as hardware ID and the compatible ID and searches a INF file with more details about the driver. If a driver as already be assigned to the device than the device should start working after plugging in. If it is the first time plugging the device, Windows Operating System uses the hardware ID of the device for other manufacturer driver and a compatible ID to search a generic driver for the device. Finally, the protection agent will prompt the user with the device requested interface after windows finding out the INF file with driver information. To the best of our knowledge, there are no free and open source solutions to address this issue for the Windows Operative Systems.

3 AGENT DEVELOPMENT

The proposed system protection agent maintained in (Oliveira et al., 2017) is based on a set of functions performed in specific stages regarding USB ports. Windows OS generates notifications for every action on USB ports and thus, this notifications can be captured by overriding a method called *WndProc* which receives a number as a code for the event. As a new device arrives, the system searches for it comparing the results of Device Manager to an old list of devices found in order to find out a new device added.

The protection agent also needs to separate working devices from the pending devices, usually, while getting the class of the device, if the class is null it will throw an exception since it has not been assigned yet. In this case, the device is probably pending and requires a driver. Devices who returned error are later added to a list of pending devices. The list of pending devices also needs a filter to determine what type of devices are relevant and irrelevant.

Although windows device manager separate devices according to their class as seen in Figure 1, the proposed protection agent selects specific USB devices by filtering devices by their hardware ID (i.e. the three first characters of the hardware id should be USB or HID).

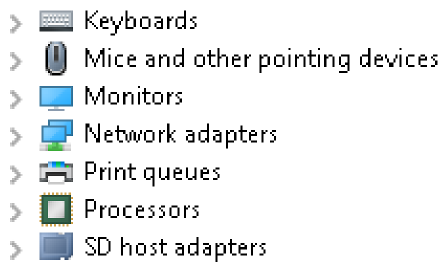


Figure 1: Device manager separating devices according to functionality.

For driver installation the system scans two separate folders where windows stores the inf files containing the driver information. These inf files can be opened as a txt file and are scanned in search for specific keywords that help us determine what interface is the device asking. By searching the driver details instead of just installing it, allows the end user to know what type of device is connected to the computer.

In Figure 2 it is presented an overall schema of the three stages used by our agent proposal. First the agent will block the device installation on Registry, then it will perform device detection and last, it will install the driver for a specific device manually. These stages are detailed in the next sections.

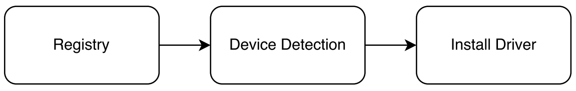


Figure 2: Functional Steps Performed by Protection Agent.

3.1 Registry

The Registry is a system database where system components and some applications retrieve stored information about configuration data (Fisher, 2017). This information varies according to the version of Windows OS. The registry stores information about user preferences, configurations for the operating system, software, hardware devices and many others. As Windows OS modifies the list of installed devices, software or any other object on the system, new values on registry will be added, edited or even removed. Windows OS installs drivers automatically without asking for user permission, this is a vulnerability that some malicious devices use to get inside Windows OS and inject malicious code.

If the host has a Pro version of Windows OS, there is an option to prevent the installation of Plug and Play devices in the Local Group Policy Editor. We can mimic that procedure by adding a specific key to the registry that denies device installation and using this agent instead to install drivers manually with a more user friendly interface. This originates a new problem, by blocking installation of all devices we are not just blocking USB devices but all types of devices. To solve this, we implemented a GateKeeper which is a simple mechanism that works basically with the same logic as the GateKeeping Theory. The GateKeeping Theory, originally created by Kurt Zadek Lewin (1890-1947), a Psychologist and pioneer in Social Psychology, was first time used to describe a wife buying food, selecting what food should end up on the dinner table and what should not (Roberts, 2005). Most commonly adopted by the media that determines what information should be displayed to the audience and what should not, this information can be from different types such as policy, religion and so on.

Figure 3 presents the gatekeeping theory, where from the list of total items some are selected and others rejected.

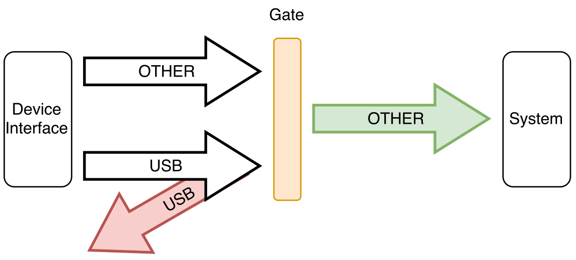


Figure 3: GateKeeping Theory, from the list of total items to the selected ones and the rejected.

3.2 Device Detection

The applications on Windows OS can capture system notifications if it registers itself using the method `RegisterUsbDeviceNotification`, this way we are able to know when a device is plugged in and the system protection agent can proceed scanning the device. Although we detect every device arrival, we do not need to block those which do not communicate over USB protocol; it was used the GateKeeping Theory.

We used Gatekeeping Theory explained in previous section to solve the issue of the previous registry key that was blocking the installation of all devices and made the system protection agent install automatically devices by scanning the hardware, finding the ID and comparing the resulting string value with USB and HID.

3.2.1 WndProc

For every action that occurs on USB ports, windows generates a notification that can be captured by overriding a method called *WndProc* which is an application-defined function that processes messages set to a window and receives a code number for the event. Multiple code numbers are specified such as code number `0x0007` or `DBT_DEVNODES_CHANGE` obtained when device added or removed, code number `0x0219` or `WM_DEVICECHANGE` used to notify a change of hardware configuration of a device or the computer, and code number `0x8000` or `DBT_DEVICEARRIVAL` obtained when a new device gets plugged in.

When a new device is plugged in, the system searches for it comparing the results of Device Manager list of devices with an old list of devices found, should a device repeat itself, the loop will skip the iteration until it finds a new device and if it was actually plugged in or out. In other words, this protection agent does not fetch a new device, it simply verifies a difference between connected devices and, after the *WndProc* function is called, the new list of connected devices.

3.2.2 Device Enumeration with ManagementObjectSearcher

`ManagementObjectSearch` collects management objects based on a query and is one of the most used to retrieve management information. It can be used, as an example, to enumerate the disk drives, network adapters and other types of hardware, processes and many other management objects on Windows. After instantiated, an instance of this class uses a WMI query (Microsoft, 2017a).

On context, a `ManagementObjectSearch` instance will be used to get all devices connected to Windows by using the `Get()` method. When this method is invoked, `ManagementObjectSearcher` runs the query and returns the collection of management objects which should contain a list of all devices on the system.

An instance of `ManagementObjectSearcher` can be instantiated with different constructors but, on this case, it only required a value of type string which contained the query to get all the devices.

3.2.3 Windows Management Instrumentation (WMI)

WMI (Microsoft, 2017c) is an infrastructure that allows data management as well as other operations such as task automation for windows based operative systems. According to Microsoft, WMI can be used in all Windows-based applications and is very useful in administrative scripts or even enterprise applications . The system is using WMI to retrieve data from the devices using a specific class called *win32 pnpentity*. This class represents all the information of a plug and play device and inside this class is a variable which should tell the interface of the device.

In the Figure 4 it is presented a system internal flow diagram. For each device that this system is able to retrieve, a new object is created to save all the information of the device. The class used for this object contains information about the device, such as class, name, ID and others which, in case of the blocked device, should appear empty and only have the *hardwareID* and the *compatibleID*.

3.2.4 Searching for INF

The system protection agent uses a simple function to search the correct inf for the captured device. It scans

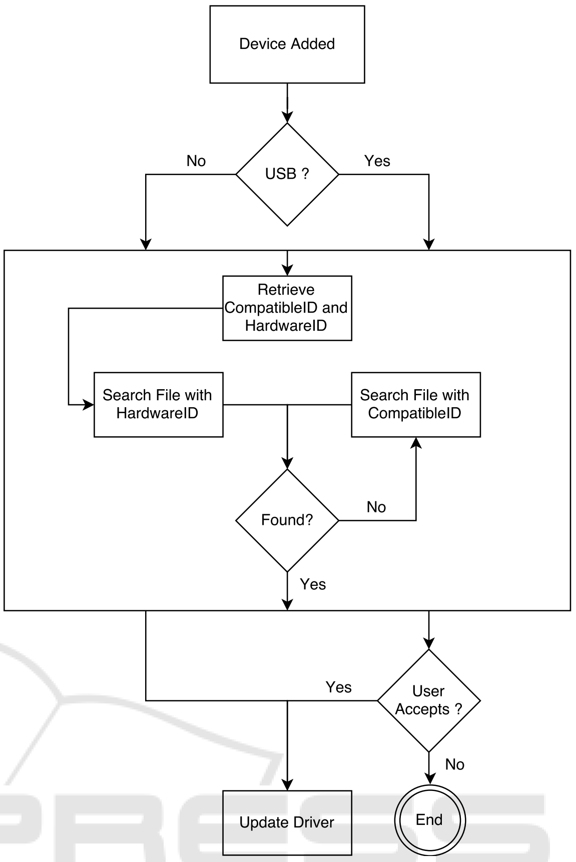

Device Added
USB ?
No
Yes
Retrieve CompatibleID and HardwareID
Search File with HardwareID
Search File with CompatibleID
Found?
No
Yes
User Accepts ?
Yes
No
Update Driver
End

Figure 4: System internal flow diagram.

two different folders:

1. `INF`
2. `FileRepository`

The first folder is where the agent starts scanning first, opening any folders inside and searching for all the files with extension ".inf". For every file with this extension, the system will read all the lines and check for the hardware ID or the compatible ID of the device. Once the system detects a match between the hardware ID or compatible ID of the device with a peace of the line, meaning that the line contains a string with one of those IDs, it adds the file full path to a list of strings that stores every compatible INF location.

In order to find the true identity of the device, this system protection agent searches on those possible INF files for specific keywords such as `SvcDesc` which should help determine the identity of the device. As an example, a thumb drive should be recognized as a USB storage device according to the information obtained from this line:

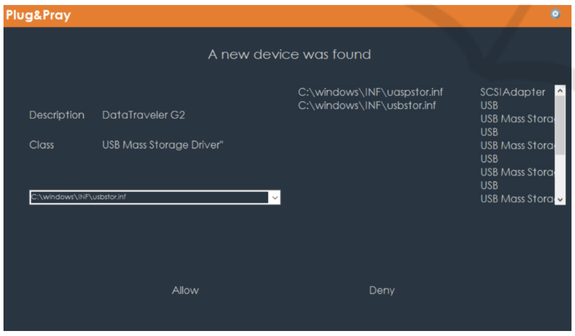
```
USBSTOR.SvcDesc = "USB Mass Storage
Driver"
```

3.3 Install Driver

When in the stage of installing a driver, there is a function called `UpdateDriverForPlugAndPlayDevices` that given a INF file and the hardware ID of a device this native function will install a driver for certain device that uses USB protocol to communicate (Microsoft, 2017b). In the current proposal, the function is called once the user accepts the device plugged in after receiving information about interface. Whether it is a keyboard, mouse or any other type of device, the previous result (see Section 3.2 and Figure 4) should return enough information about the device that the user is trying to connect to the Windows Operative system.

4 VALIDATION TESTS

The proposed protection agent was built and deployed in a Windows 10 Pro Creators Update, version 10.0.15063. After the execution of the protection agent the same was tested against many devices such as thumb drives, keyboards, mice and pointing devices, network adapters and other types of USB devices. When plugged in, if allowed on the settings panel, all tested USB devices triggered the graphical interface presented in Figure 5 enabling the user to choose between the USB devices that are being blocked in the system.


Plug&Pray

A new device was found

Description DataTraveler G2

Class USB Mass Storage Driver"

C:\windows\INF\uaspstor.inf
C:\windows\INF\usbstor.inf

SCSIAdapter
USB
USB Mass Stora
USB
USB Mass Stora
USB
USB Mass Stora
USB
USB Mass Stora

C:\windows\INF\usbstor.inf

Figure 5: Device SvcDesc and INF location.

The proposed system protection agent was also tested using an Open Source Android penetration testing platform, capable of HID keyboard and BadUSB Man-in-the-middle attacks, namely the Kali Linux Nethunter (kali.org, 2017). As result, the agent succeeded in recognizing the pseudo device as a keyboard, blocked the device and requests from user further instructions.

The performance of the current proposal depends if the system protection agent finds a match between hardware ID or compatible ID of the device on the folder with the INF files. According to the OS version, the amount of devices connected prior to the first launch of this system protection agent, it can take some milliseconds to find a match because the total number of INF files will probably not be the same in every computer. Printing the current file path to the console while scanning the files will also result on a delay between device connection and possible class notification and therefore the output on the console is only used for testing purposes.

5 CONCLUSIONS AND FUTURE WORK

Security attacks using USB interfaces and devices impose high risks to users and companies. In particular the BadUSB attack allows attackers to inject malicious scripts into a target OS.

This paper proposes a protection agent against BadUSB attack developed for Windows operative systems. It allows a user to check the class of an USB device ready to be mounted, though enabling the detection of a potential attack if the expected functionality of the device does not match with its class type.

The results show that the proposed protection agent is capable of detecting potential intrusions by blocking the installation of the device, scanning the device for something that identifies it, searching for a description locally and finally warning the user about the device meaning that all devices must be approved by the user when plugged in if the system protection agent is running.

In spite of the protection provided by the proposed system agent, identifying and intercepting the installation of the devices drivers, there are limitations that should be addressed as future work. There are some performance issues that should also be improved and an effective filtering should also be deployed for multiple USB devices. User interaction with the proposed system agent can also be improved in order to facilitate the user decision. The current implementation still relies on the user knowledge to allow or block the device installation and thus, there is the real risk that malicious devices are wrongly accepted, potentially harming the system.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their detailed comments that helped to improve this paper.

The authors acknowledge the support by

Fundação para a Ciência e a Tecnologia (FCT) through the Computer Science and Communication Research Centre (CIIC), Portugal.

REFERENCES

Andreasson, K. J. (2011). *Cybersecurity: public sector threats and responses*. CRC Press.

Benchoff, B. (2013). Extrating data with USB HID. Website accessed on 2017-10-10 (https://hackaday.com/2013/01/26/extracting-data-with-usb-hid/).

Cannon, T. (2010). Data leak prevention bypass. Website accessed on 2017-10-10 (http://thomascannon.net/dlp-bypass/).

Cheng, N. (2014). Protect against badusb with mobile cloud storage. Website assessed on 2017-10-06 (https://chaione.com/blog/protect-badusb-cloud-storage/).

Crenshaw, A. (2011). Plug and prey: Malicious usb devices. *Proceedings of ShmooCon*.

Fisher, T. (2017). What is the Windows Registry? Website assessed on 2017-10-06 (https://www.lifewire.com/windows-registry-2625992).

Greene, T. (2010). Conficker worm takes Manchester police offline for three days. Website assessed on 2017-10-06 (http://www.computerworlduk.com/security/conficker-worm-takes-manchester-police-offline-for-three-days-18640/).

Jodeit, M. and Johns, M. (2010). Usb device drivers: A stepping stone into your kernel. In *Computer Network Defense (EC2ND), 2010 European Conference on*, pages 46–52. IEEE.

kali.org (2017). Kali linux nethunter for nexus and oneplus. Website accessed on 2017-10-13 (https://www.kali.org/kali-linux-nethunter/).

Kamkar, S. (2016). PoisonTap – exploiting locked computers over usb. Website accessed on 2017-10-10 (https://samy.pl/poisontap/).

Microsoft (2017a). ManagementObjectSearcher. Website assessed on 2017-10-06 (https://msdn.microsoft.com/en-us/library/system.management.management-objectsearcher(v=vs.110).aspx).

Microsoft (2017b). UpdateDriverForPlugAndPlayDevices function. Website assessed on 2017-10-08 (https://msdn.microsoft.com/en-us/library/windows/hardware/ff553534(v=vs.85).aspx).

Microsoft (2017c). Windows Management Instrumentation. Website assessed on 2017-10-06 (https://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx).

Oliveira, J., Pinto, P., and Frade, M. (2017). usb-whitelisting. DOI:10.5281/zenodo.1009691 https://doi.org/10.5281/zenodo.1009691.

Roberts, C. (2005). Gatekeeping theory: an evolution. Website accessed on 2017-10-13 (http://www.reelaccurate.com/about/gatekeeping.pdf).

Sood, A. and Enbody, R. (2014). *Targeted cyber attacks: multi-staged attacks driven by exploits and malware*. Syngress.

Symantec (2017). Endpoint Protection - Machine Learning Security | Symantec. Website assessed on 2017-10-06 (https://www.symantec.com/products/endpoint-protection).

TechGuru (2017). Reviews of the best USB keyloggers. Website accessed on 2017-10-10 (https://nerdtechy.com/reviews-best-usb-keyloggers).

Terdiman, D. (2012). Stuxnet delivered to Iranian nuclear plant on thumb drive. (Worm). Website assessed on 2017-10-06 (https://www.cnet.com/news/stuxnet-delivered-to-iranian-nuclear-plant-on-thumb-drive/).

Tian, D. J., Bates, A., and Butler, K. (2015). Defending against malicious usb firmware with goodusb. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 261–270, New York, NY, USA. ACM. DOI:10.1145/2818000.2818040, (http://doi.acm.org/10.1145/2818000.2818040).

Ulanoff, L. (2014). How you can avoid a badusb attack. Website assessed on 2017-10-06 (http://mashable.com/2014/10/03/how-can-you-avoid-badusb/).

USBKill.com (2017). USB kill v3. Website accessed on 2017-10-10 (https://usbkill.com/products/usb-killer-v3).

Veres-Szentkiralyi, A. (2012). Leaking data using DIY USB HID device. Website accessed on 2017-10-10 (http://techblog.vsza.hu/posts/Leaking_data_using_DIY_USB_HID_device.html).

Walters, P. (2012). The Risks of Using Portable Devices. In *United States Computer Emergency Readiness Team*.