International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 – 5161

Available at http://inpressco.com/category/ijcet

Research Article

Unintentional use of USB Channels and Protection

Pravin Phule†* and Rekha Kulkarni†

†Department of Computer Engineering, PICT, Pune, Maharashtra, India

Accepted 30 Jan 2015, Available online 05 Feb 2015, **Vol.5, No.1 (Feb 2015)**

Abstract

Protecting an organization against the information leakage is a big task, especially when the attacker is an insider. This paper covers an insider attack based on Unintentional use of USB channels. Endpoint Protections are available to guard against the leakage of information over USB Mass Storage and USB Printer interfaces but they fail to keep watch on flow of data over USB Human Interface Devices such as USB speaker, USB keyboard and USB mouse etc. Operating system allows USB device to identify itself as any USB device. To exploit this vulnerability a Hardware Trojan horse device is developed which can identify itself as USB audio device or USB Keyboard device and make Unintentional use of USB channels to steal the data. This paper also gives a possible protection scheme to detect and block the stealing of data over Unintended USB Channels.

Keywords: *Hardware Trojan horse, Protection, USB Channels, USB Security*

1. Introduction

USB devices have now become the most reliable and most frequently used type of devices. Almost every purpose is satisfied with an appropriate USB device-storage (flash drive), printing (printer), typing (keyboard); dimensional input (mouse) or audio (speaker) etc. All this has become possible because USB specification provides a single physical interface with base protocol for all USB devices. Whatever may be the type of USB device interface remains same (USB Implementers Forum, 2001). Ease of USB device lies in its Plug and Play feature that signifies no need to install the device on computer system with some kind of software driver, just plug it and use. Plug and play feature makes USB device more favorable. But this feature also leads to the risk of malware and Trojan attacks (Stasiukonis, 2006).

To deal with plug and play protective software solutions are available to keep your system safe and block the information leakage through plug and play (Centenial Software, 2012), (CheckPointSoftware, 2009), (Device Lock Inc., 2014). Endpoint security solutions (ESSs) for a corporate can be employed to govern the flow of information from computer system to the attached devices so that there is no leakage of information. ESSs can allow certain users to access certain devices (but not all the devices) inside the organization by enrolling their serial numbers in the Access Control List (ACL). This strategy works, when only the ACL enrolled devices are permitted for inside use. But ESSs cannot guarantee the same in case of USB Human interface devices like mouse, keyboard, headsets or speakers etc., (Centenial Software, 2012), (CheckPointSoftware, 2009), (Device Lock Inc., 2014). This breach into security can be exploited with a development of a Hardware Trojan horse Device to make an unintentional insider attack to the computer system in an organization to steal the information of interest. Unintentional use of USB channel means that, using USB channel for doing something which is not signified in USB specification (Clark, *et al.,* 2009), (Clark, *et al.*, 2011). Let's take an example of communications between a USB keyboard and a computer. There are two channels between USB Keyboard and computer system for intentional use. One for the transmission of key presses from keyboard to computer and other for the transmission of keyboard LEDs status (Caps Lock, Num Lock and Scroll Lock) from computer to USB keyboard. An application running on the computer system could make an unintentional use of USB keyboard channel to transfer the information of interest in the form of toggling keyboard status LEDs (Clark, *et al.*, 2009), (Clark, *et al.*, 2011). Similarly unintentional use of USB Audio Channel to transfer the information of interest in the form of audio packets is also possible.

2. Related Work

As per the USB specification operating system allows the attached USB device to identify itself as any kind of USB device. Using this subjection a USB Meta-Device can be programmed to enumerate itself as any kind of

*Corresponding author: **Pravin Phule**

USB device with a mischievous driver loaded on the computer system (D. Barral and D. Dewey, 2005).

The permitted use of USB devices can be exploited using USB Auto Run and Auto Play features to copy data of a computer system and automatically execute malicious code on a network endpoint without the user's knowledge (M. Al-Zarouni, 2006). A software Trojan horse can be developed using covert channels of various network layers. Unintended USB channel is same as covert channel though we don't put any extra effort to make it unobservable (Shah, *et al., 2006*). It can be used between the keyboard and computer as a key logger. It can also be used to pull out the data from a computer system by adjusting the timing information details of packets transferred over the internet. However, it requires that the computer with internet to maintain the session. In our project there is no need of internet connectivity to pull out the information of interest from the computer system.

Use of less reliable integrated circuits (ICs) may lead to the process reliability Trojans (Shiyanovskii, 2010). Due to the high cost of new fabrication facilities, industries are choosing the less reliable integrated circuits(IC). Vulnerabilities of such integrated circuits (ICs) can be exploited to make a hardware Trojan and there are some ways to detect it (Ronald Smith and Scott Knight, 2008). Digital IP cores from the third party cannot be assumed as trusted because a Trojan can be inserted into them (Xuehui Zhang and Tehranipoor, 2011). However, in this project we are not focusing on less reliable ICs or untrusted IP cores. But this paper focuses on the liability of the operating system which relies on USB device for its identification. The device may identify itself as any kind of USB device such as USB keyboard or USB speaker and communicate over Unintended USB audio channel.

Vulnerable keyboard controller can be maliciously used to gain the passwords using the noise mingling technique even in presence of the keyboard protection software. However, the attacker can be blocked from gaining the exact password (Kangbin Yim and Soonchunhyang, 2010). Another noise mingling technique is discussed in (Lee, *et al., 2010*). There is a solution that can be applied to the vulnerable keyboard controller. But, in this paper we have focused on the possible solution to block unintentional use of USB channels.

The Hardware Trojan horse device for the unintentional use of USB Channels is implemented in (Clark, *et al.*, 2009), (Clark, *et al.*, 2011). The Hardware Trojan horse device is built using PLXs Net 2280 Programmable Peripheral Controller (PLX Technology, 2008) which is an expensive development platform. We have used a low cost USB prototyping device for the development of Hardware Trojan horse device for the unintentional use of USB Channels: Mbed NXP LPC11U24 Microcontroller (Mbed, 2012). Also, in this paper we have implemented the possible solution to block the exfiltration of data over unintended USB channels.

3. Problem Definition and Solution

The concept of Hardware Trojan horse device (Clark, *et al.*, 2009), (Clark, *et al.*, 2011) and the solution is discussed below.

3.1 Problem Definition

An application on the computer system could create an unintended USB keyboard channel to pull out the information of interest in the form of toggling keyboard status LEDs. Also an application on the computer system can be used to send the confidential data over unintended USB audio channel by using Isochronous Out Transfer line ((USB Implementers Forum, 2001) and send the confidential data by putting it in audio packets and playing the audio file. Fig.1 shows Interrupt In, Control Out and Isochronous Out lines of communication between USB Device and computer system.

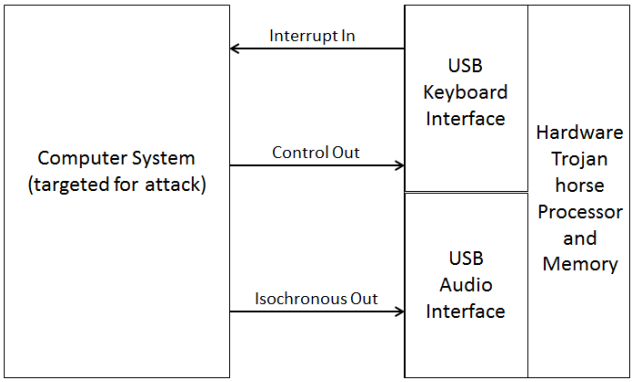

Fig.1 Hardware Trojan horse Device

The Human Interface Devices: USB keyboard and USB speaker are targeted in this project. According to previous research (Clark, et al., 2009) and (Clark, et al., 2011); Human Interface Devices: USB keyboard and USB speaker are not well regulated by Endpoint Security Solutions. But ESSs do regulate the devices such as USB flash, USB printer etc.

3.2 Protection Software (Solution)

The solution can be in the form of software application on the computer system. This software will keep a watch on USB keyboard channel by detecting continuous toggling of Keyboard status LEDs and USB audio channel by recording audio being played for specific pattern of data in audio packets.

After detecting such type of activity, it will be automatically blocked (Pravin Phule, 2012). But with this approach there is a chance of exfiltration till it gets detected. Therefore the exact source of exfiltration needs to be blocked. The source of this exfiltration lies in the specific pattern observed in the files used for exfiltration. The files with suspicious patterns that may cause the exfiltration of data by making Unintentional use of USB Channels must be immediately removed.

Implementation of the Protection Software based on detection and removal of files with specific suspicious patterns is discussed in more detail in section 5.2. Fig.2 shows the solution regulating the flow of data over Control Out and Isochronous Out lines.

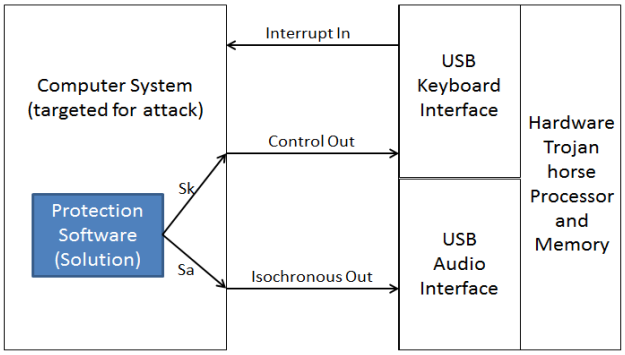

Fig.2 Protection Software (Solution)

4. Unintentional Use of USB Channels

In this research we are considering USB channels which are not well regulated by the Endpoint Security Protections. These USB channels are classified into two types: User-Space and Kernel-Space channels (Clark, et al., 2009), (Clark, et al., 2011). In case of User-Space channels, the USB protocol is enough to make unintentional use of USB channels; but in case of Kernel-Space channels, the computer system's kernel can be disrupted and the USB System Software can be modified to make unintentional use of USB channels. In this paper we focus only on User-Space unintended USB channels.

There are four types of USB transfers: Bulk Transfer (used by USB flash drives), Control Out (used by the operating system to send LED status to the all attached keyboards), Interrupt In (used by USB keyboards to send the key presses to the computer system) and Isochronous Out (used by the operating system to send audio packets to the USB Speaker when an audio file is played). Bulk transfers are protected with the use of Endpoint Security Solutions. Interrupt transfers are not considered because they send the key presses from USB keyboard device to the computer system, therefore Interrupt In cannot be used to pull out the information of interest from computer system. Control Out and Isochronous Out transfers are only considered for this research. As per the need of this project, Hardware Trojan must be capable of identifying itself as USB keyboard or USB speaker during the enumeration process. Almost every computer requires the keyboard. The Endpoint Security Solutions don't consider it for regulating the flow of data. Consequently, the Control Out transfer to the USB keyboard can be used unintentionally to pull out the information of interest. USB keyboards are Low-Speed devices. Higher throughput can't be expected from unintentional use of USB keyboard channel. For a good, considerable throughput the Isochronous Out transfer to the USB speaker is used. As USB speaker can only receive audio packets, it is not considered by Endpoint Security Solutions for regulating the flow of data.

4.1 Unintentional use of USB Keyboard Channel

As compared to USB speaker, USB keyboard is a Low Speed Human Interface Device. There are two channels between USB Keyboard and computer system used intentionally: Interrupt transfer for the transmission of key presses from keyboard to computer and Control Transfer for the transmission of keyboard LED status (Caps Lock, Num Lock and Scroll Lock) from computer system to USB keyboard. Thus Control Transfer line of USB Keyboard can be used to pull out the information of interest from computer system to the attached USB keyboard.

Table 1 Keyboard Output Report (USB Implementers forum, 2001)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved			Kana	Compose	Scroll Lock	Caps Lock	Num Lock

As given in the USB Specification (USB Implementers forum, 2001) and the USB Device Class Definition for Human Interface Devices (USB Implementers forum, 2001) the format of Keyboard Output Report used by the Control Transfer is given in Table 1. Computer system generates the Keyboard Output Report each time the keyboard transfers the modifier key press to computer system.

Keyboard Output Report is transmitted by computer system to all attached keyboards. Using the Keyboard Output Report, the attached USB keyboards toggle their corresponding LEDs. The three modifier bits from the Keyboard Output Report can be used for the exfiltration of data. Keyboards Output Reports can be generated after every 109.5 milliseconds. That indicates the exfiltration of 3 bits of information after every 109.5 milliseconds. The theoretical throughput is 3.42 bytes/sec.

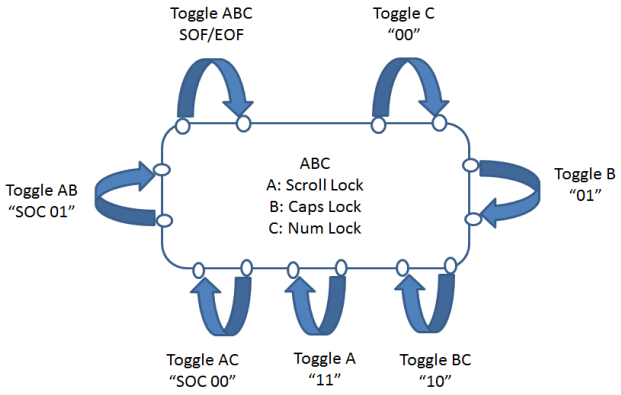

Fig.3 Keyboard LED Coding Scheme (Clark, et al., 2009)

Keyboard LED coding scheme used in this project is based on simple state diagram as shown in Fig.3. The

Keyboard LED Coding Application on the computer system can make the use of status modifiers (Scroll Lock, Caps Lock, and Num Lock) to encode the information of interest in the form of toggling keyboard LEDs. It encodes the data of input file in the form of toggling keyboard LEDs using a VBScript file and that VBScript file is executed. In this coding scheme seven symbols are used to represent 7-bit ASCII characters. In Fig.3 we see that the state machine can begin at an arbitrary state to toggle the status modifier keys, ABC: where A, B, C contain the current states of Scroll Lock, Caps Lock and Num Lock respectively. A 7-bit ASCII code can be generated using a Keyboard Output Report from which one, two or all the three LED bits are toggled. In these seven symbols, four symbols represent 2-bit information; two symbols represent 1-bit information (padded with zero on the left), which also represent Start of Character and the remaining one symbol represents Start/End of Frame (SOF/EOF). Toggling of SOF/EOF symbol twice indicates start and end of a message which creates overhead of 1 byte for the message of any arbitrary size. To exfiltrate a single character, four Keyboard Output Reports are generated. For example, to exfiltrate character "R" (7-bit ASCII: 0101 0010) the following four Keyboard Output Reports are generated: Toggle AB (Scroll Lock and Caps Lock) (SOC 01); Toggle B (Caps Lock) (01); Toggle C (Num Lock) (00); and Toggle BC (Scroll Lock) (10).

Keyboards Output Reports can be generated after every 109.5 milliseconds. This delay is imposed to ensure that Keyboard Output Report is successfully received by the USB keyboard interface (Clark, et al., 2009). For a message of size n bytes, the Keyboard LED Channel Throughput (Clark, *et al.*, 2009) can be calculated as follows:

$$Throughput = n/\{4 * (n + 1) * 0.1095\} \quad (1)$$

The theoretical throughput of 3.42 bytes/sec can never be achieved with the chosen LED coding scheme due to the overhead of framing characters. Because of the imposition of overhead by the framing symbols (SOF/EOF), the maximum throughput that can be achieved is about 2.283 bytes/sec.

4.2 Unintentional use of USB Audio Channel

Unlike USB keyboard, USB speaker is a Full-Speed USB audio device. USB audio device uses two types of USB Transfers with the computer system: Control Transfer which is used to communicate the control data with the computer system; the other is an Isochronous Transfer which is used to receive the audio packets from the computer system. The USB Specification specifies the maximum size of data packet using Isochronous Transfer as 1023 bytes. Also a Full-Speed device can receive a data packet of size 1023 bytes every millisecond which indicates the throughput of 1023 kB/s.

The encoding style outlined in the USB Device Class Definition for Audio Devices 2.0 (USB Implementers forum, 2006), is used for the transmission of isochronous data to an audio device into structured blocks. The size of each audio block in bytes/sample can be given as follows:

$$\text{Block Size} = \text{Number of Channels} * \text{Sample Resolution} \quad (2)$$

As we are using MBED LPC11U24 microcontroller, a low speed, low cost USB prototyping device, we cannot expect the Full-Speed audio interface on the device. We have chosen the audio coding scheme to match with the audio interface that works well with the device. The PCMWAVEFORMAT (Microsoft, 2014) audio format is used in this project with sample resolution of 2 bytes/sample and one channel. That means we get a block size of 2 bytes. The sample rate of the audio file is also responsible for the number of data blocks that can be embedded into one audio packet. The PCMWAVEFORMAT audio format is used with a sample rate 48000 samples per second for each channel. That means 48 blocks of 2 bytes, per data packet are transmitted every 1 millisecond. With these details for an audio file we can calculate the size of an audio packet as follows:

$$\text{Audio Packet Size} = \text{Number of Blocks} * \text{Block Size} \quad (3)$$

Therefore the audio packet size is 48 * 2 = 96 bytes. Thus with PCMWAVEFORMAT, we get the Theoretical Throughput of the Audio Channel as about 96 kB/s.

To ensure Theoretical Throughput we used Teraterm Terminal Emulator (Sourceforge projects, 2013) to observe audio data that is being sent to the device. We developed the USB Audio Coding Application in C# which allows us to select the text file to embed the data of text file into the audio packets and create a wave files according to the above mentioned format. Also the USB Audio Receive interface, to receive and store audio packets, for the device Mbed LPC11U24 was developed in C++ using Mbed Online Compiler (Mbed, 2013). Wave files created using USB Audio Coding Application were played using PlaySound() API and Windows Media Player. After the series of experiments, the observed obstacles and solutions to cope up with them are as follows:

1) Change in audio representations

For representing every ASCII character embedded in the audio file there are 3 equivalent audio codes observed when the audio is played (Clark, *et al.*, 2009), (Clark, *et al.*, 2011). But in our project we observed up to 8 audio codes for the ASCII character. Therefore we considered a single audio code and its nearby values for decoding an audio code to its equivalent ASCII character. We also observed that these audio codes change after an unidentified period of time but not immediately. So we programmed USB Audio Coding Application as follows: USB Audio Coding Application

first creates and plays the wave files which are embedded with the ASCII characters. After that similarly it embeds the text file's data in a wave file and plays it over the USB Isochronous Transfer line.

2) Size of data

We observed that exfiltrated audio codes get disturbed when the full size (96 bytes) data is embedded in an audio packet. We tried for different values for the size of data to be embedded in an audio packet and then came to the conclusion that the data size should be 47 bytes. USB Audio Coding Application was reprogrammed so that it embeds 47 bytes of data repetitively in an audio packet till it reaches the maximum value i.e. Audio Packet Size: 96 bytes. After that it was observed the consistent audio codes for each character which were played in the form of audio over the USB Isochronous Transfer line.

3) Type of Characters

Due to the observed inconsistency in audio codes, we were not able to map the lowercase letters and other remaining symbols to their equivalent audio representations. Because of the variability in the data transmitted, we decided to use only uppercase ASCII letters, numbers and some punctuation symbols. Total 68 ASCII characters were used for the transmission of data in the form of audio packets. Each of the transmitted symbols can be mapped to its equivalent audio representations after it has been played.

4) Receiving capability of device

On the other hand, we also observed that Mbed audio device receives and stores the inconsistent audio codes when we tried to save the full 96 bytes packet on the device. Then we tried to save the half of the audio packet and after that we found that consistent audio codes were stored on the device. So we programmed the USB Audio Receive interface on the device so that when it receives an audio packet, only half of the audio packet is stored on the device.

5) Size of wave

It was observed that when we played a small sized wave file, nothing was stored on the device. After that we observed that when we played a wave file which is at least 5 seconds long in duration then only the half of the audio packet (48 bytes) is stored on the device. Already we made the conclusion for the size of data to be embedded in an audio packet as 47 bytes. So we modified USB Audio Coding Application in such a way that for every 47 bytes of data it creates a separate wave file by embedding the data repetitively in audio packets so as to make it of at least 5 seconds long in duration. Also we kept a digital silence of 2 seconds after each file is played so that audio device successfully receives and stores the audio data.

6) Ordering

The audio data packet stored on the device was not in the ordered sequence making it difficult to decode. It was observed that if the first audio code stored on the device whose equivalent ASCII character was at odd location in played audio packet, then it was followed by the audio code whose equivalent ASCII character was found at the next odd location in the played audio wave file. That means if first byte is odd then it will be followed by next odd bytes which does not mean that first byte stored on the device whose equivalent ASCII character code is always at the first location in played audio packet. Finally we made conclusion that odd bytes will be followed by the next ordered odd byte and this continues up to the last odd byte in the audio packet; next to that all even bytes are stored; followed by the remaining odd bytes if any, starting from the first odd byte. The same pattern was observed for even bytes. That means equivalent audio code for the first ASCII character in played audio packet can be at any location in the data packet stored on the device. So we used starting delimiters, ASCII character '>' to indicate the start of odd bytes and '?' to indicate the start of even bytes.

7) Security Restriction

Upload Application to reside on Mbed LPC11U24 device as an USB Keyboard interface is discussed in section 5.1. This application can be used to upload either Keyboard LED Coding Application or USB Audio Coding Application in the form of key presses to the computer system. For this we converted both of the Keyboard LED Coding Application and USB Audio Coding Application from exe to VBScript using Exe2VBS binary encoder as used in previous research (Tarasco, 2003). VBScript file contains ASCII characters that can be converted into key presses. Upload Application converts ASCII characters from VBScript file into their equivalent key presses and send them to computer system. This method was worked well on Windows XP (Clark, *et al.*, 2009). Exe2VBS binary encoder just wraps the exe file into VBScript file. After executing such VBScript file, it first, unwrap an exe file and then that exe file is executed. But now an exe (or any other file) file generated from VBScript file is identified as a Trojan and immediately gets quarantined before it executes (McAfee Inc., 2014). Therefore we decided to upload the C# code files without any conversion in the form of key presses. We chose three files from C# project for uploading in the form of key presses: Form1.cs, Form1.designer.cs and Form1.resx. The Keyboard LED coding application also makes the use of VBScript file for toggling keyboard LEDs. But it uses the plain VBScript code instead of wrapping exe inside it. It implies that plain VBScript code can directly execute and it does not get caught by the antivirus protections.

After considering all the above discussed things, it is clear that we can exfiltrate 47 bytes in a period of 5

seconds and 2 more seconds are required to put the gap between two consecutive exfiltrations. That means we are exfiltrating 47 bytes per 7 seconds. In those 47 bytes, 2 bytes are used to indicate the start of odd and even bytes; and remaining 45 bytes are for data. Throughput of exfiltration is 45/7= 6.42 bytes per second. It is not that high but definitely it is higher as compared to the exfiltration throughput of keyboard LED channel (between 2 to 3 bytes/s). A consistent throughput can be achieved when PlaySound() API is used for playing wave files, instead of Windows Media Player. We are considering both the PlaySound() API and Windows Media Player for the analysis of exfiltrated data. The highest possible throughput by considering all the methods of exfiltration implemented in this project is not more than 6.42 bytes per second. Because no matter whatever is the throughput, our ultimate aim is not only to develop a Hardware Trojan horse device based on Unintentional use of USB Channels but also to find the exact source that is responsible for the exfiltration of data by making Unintentional use of USB Channels with the probable solution to block them.

We know that USB speaker is a high speed device, but still when we play the audio we are able to store only the half of audio packet within duration of 5 seconds. The obvious reason behind this is that when we are using Mbed LPC11U24 device as an USB speaker, it is only intended for playing the audio. But still we are able to catch half of the audio packet within duration of 5 seconds. Even this half packet is sufficient to make the unintentional use of USB Audio Channel.

5. Implementation

Design specific details for this project such as the Mathematical Model, Objects identified in the Project, Morphism, Overloading functions, Implementation methodology and Feasibility Assessment are already discussed in our previous research papers (Pravin Phule, 2012), (Pravin Phule, 2013). Preliminary results of implementation can be found in (Pravin Phule, 2013). In this paper we have focused on the final implementation of a low cost Hardware Trojan horse device based on unintentional use USB channels and the probable solution to block it.

5.1 Development of Hardware Trojan horse Device

Development of Hardware Trojan horse can be divided into 6 modules (Pravin Phule, 2012), (Pravin Phule, 2013):

1) Keyboard LED Coding Application:
It is a malicious code used to send the information of interest to the attached device by making unintentional use of USB keyboard channel in the form of toggling keyboard LEDs.

Keyboard LED Coding Application was developed in C# which can encode the input text in the form of toggling keyboard LEDs and write the equivalent Keyboard LED toggling sequence in the VBScript file and the VBScript file is executed. USB Keyboard Receive interface on the device continuously receives the status of keyboard LEDs with a gap of 109.5 milliseconds. Based on the received Keyboard Output Reports, the data is decoded to plaintext characters and stored on the local storage of device. VBScript file is chosen because it provides the way to toggle the keyboard LEDs and windows allow a VBScript file to execute directly. The Keyboard LED Coding scheme used for developing this application is discussed in section 4.1.

2) USB Audio Coding Application

It is a malicious code used to send the information of interest to the attached audio device in the form of audio packets by making Unintentional use of USB audio channel.

USB Audio Coding Application is developed in C# which is able to embed the data in the form of audio packets and plays them by making Unintentional use of USB audio channel so that USB Audio Receive interface on the device receives and stores them. More information about the coding scheme used in USB Audio Coding Application can be found in section 4.2.

3) Upload Application on the device

It is the application on Mbed LPC11U24 device which will act as USB keyboard interface and upload the malicious code to the computer system in the form of key presses.

Upload Application is developed in C++ using Mbed Online Compiler (Mbed, 2013). This application can be used to upload either Keyboard LED Coding Application or USB Audio Coding Application in the form of key presses to the computer system. For this we converted both of the Keyboard LED Coding Application and USB Audio Coding Application from exe to VBScript using ExetoVBS binary encoder (Tarasco, 2013) as used in previous research (Clark, *et al.*, 2009). As discussed in section 4.2, Exe2VBS binary encoder just wraps exe file into VBScript file instead of converting it to VBScript. It first unwraps and then executes. But now such file is identified as a Trojan and immediately gets quarantined before the execution (McAfee Inc., 2014). Therefore we decided to upload the C# code files without any conversion in the form of key presses. We chose three files from C# project for uploading in the form of key presses: Form1.cs, Form1.designer.cs and Form1.resx. C# project files contain ASCII characters that can be converted to their equivalent key presses. Upload Application converts ASCII characters into key presses and then sends them to computer system. After uploading, the C# project can be executed to exfiltrate the data by making Unintentional use of USB Channels.

4) USB Keyboard Receive Application

It is the application on Mbed LPC11U24 device which will act as USB keyboard interface; which will be able to receive Keyboard LED status report from the computer system and decode the LED encoding to the characters of data. USB Keyboard Receive Application is developed in C++ using Mbed Online Compiler (Mbed, 2013).

Data to be exfiltrated is encoded in the form of toggling keyboard LEDs in a VBScript file and it is executed to toggle the keyboard LEDs. At the same time USB Key-board Receive Application receives, decodes and stores the decoded text on the device.

5) USB Audio Receive Application

It is the application on Mbed LPC11U24 device which will act as USB audio interface and it is able to store the audio packets received from the computer system.
USB Audio Receive Application is developed in C++ using Mbed Online Compiler (Mbed, 2013). When USB Audio Coding Application starts exfiltrating the data in the form of audio packets, USB Audio Receive Application on the device receives and stores them.

6) USB Audio Decode Application

It is the application required for decoding and extracting the data from the exfiltrated audio packets.
USB Audio Decode Application is developed in C# which takes input as the text file containing audio packets stored by the USB Audio Receive Application and outputs the decoded text. The more clear idea about decoding the exfiltrated audio packets can be found in section 4.2.

5.2 Development of Protection Software

Here we discuss the implementation of the Solution (i.e. Protection Software) mentioned in section 3.2. As shown in Fig.2, the Solution prevents the exfiltration of data by making unintentional use of USB Audio and USB Keyboard channels. The Protection Software can be discussed in two parts:

1) Exfiltration of data from the computer system to the attached device in the form of toggling Keyboard LEDs can be prevented as follows:

C# application is developed to keep the watch on file system looking for the specific VBScript files. Whenever a VBScript file is saved on the computer system storage, the software will check whether the file contains the code to toggle keyboard LEDs with a delay of 109.5 milliseconds between the toggling sequences. Such code may be used to exfiltrate the data and if found then the file will be deleted immediately with the message, "The file may lead to the Unintentional use of USB Keyboard Channel. Therefore it has been removed." The software deletes the VBScript file before it exfiltrates the data in the form of toggling keyboard LEDs. Therefore there is no chance of exfiltrating the data by making unintentional use of USB Keyboard Channel.

2) Exfiltration of the files from the computer system to the attached device through the USB Audio Channel can be blocked as follows:-

Once again using C# file system watcher, we implemented this part. Whenever an audio wave file is saved on the computer system storage, the software will check whether the file contains the plaintext and if found then it will be deleted immediately with the message that "The file may lead to the Unintentional use of USB Audio Channel. Therefore it has been removed." The software deletes the audio file before it plays. Therefore there is no chance of exfiltrating the data by making unintentional use of USB Audio Channel.

Both the solutions work in parallel to block the Unintentional use of USB Keyboard and USB Audio Channel.

6. Experiments and Results

This section provides performance results of data exfiltration using USB Keyboard and USB Audio Channel. It also gives the comparison between various methods used for exfiltration. We have used a low cost USB Prototyping Device: Mbed LPC11U24 Microcontroller. With the help of USB Keyboard LED Coding Application and USB Audio Coding Application, we have exfiltrated 753 bytes of data. The bulk data is not considered for this analysis, because the throughput of exfiltration is not suitable for bulk data. The selected amount of data is enough to prove the Unintentional use of USB Channels. Also we have implemented the solution which is adequate to block the Unintentional use of USB Channels.

We implemented a low cost Hardware Trojan horse Device based on Unintentional use of USB Channels, using Mbed LPC11U24 Microcontroller. A Hardware Trojan horse is just like Software Trojan; because it also provides malicious functionality in addition to that it makes itself a legitimate activity. The researchers are now considering the risks of Unintentional Insider Threats as in (Bureau, Federal Infrastructure Protection, 2013). However, as discussed in Section 3.3, most of the methods used to detect the Hardware Trojan are based on analysis of computer system's IC; which may be used to find out the time at which Trojan activates and power that it consumes (Salmani, *et al., 2009*). The Hardware Trojan implemented in this paper is different because it is in the form of USB device. It has its own chip processor. It is not dependent on computer system for its processing.

We have developed a Hardware Trojan interface as given in Fig.1 based on Unintentional use of USB Channels. The Hardware Trojan may identify itself as USB Keyboard or USB Speaker and exfiltrate the data

using Keyboard LED Channel (Control Out Transfer) and the Audio Channel (Isochronous Out Transfer).

6.1 Attack Scenario

Hardware Trojan horse device with a keyboard interface on it is attached to computer system. The Hardware Trojan horse device works like a keyboard; it can send the key presses to computer system and receive the keyboard output reports from computer system. The Hardware Trojan is operated outside the organization's regular hours. After that it uploads the malicious code required to make Unintentional Use of USB Keyboard and USB Audio Channels. Uploaded Applications execute and attacker may select the data to be exfiltrated. Then it causes the exfiltration for the information of interest from the computer system to the Hardware Trojan horse device by making Unintentional Use of USB Audio Channel. Finally, the Hardware Trojan ends its attack by returning the computer system to normal state.

6.2 Uploading Code

To upload the malicious code to computer system we have used the Upload Application. Upload Application resides on Mbed LPC11U24 device which will act as USB keyboard interface and upload the malicious code to the computer system in the form of key presses. Upload Application is developed in C++ using Mbed Online Compiler (Mbed, 2013). This application can be used to upload either Keyboard LED Coding Application or USB Audio Coding Application in the form of key presses to the computer system. Hardware Trojan is setup to act like USB keyboard; it can send the key presses to the computer system. Following files were uploaded to a target computer system, in the form of key presses generated using the Hardware Trojan's keyboard interface:

1) Keyboard LED Coding Application's C# Project Code files:

Form1.cs, Form1.designer.cs and Form1.resx; Keyboard LED Coding Application encodes the data to be exfiltrated in the form of VBScript code for toggling keyboard LEDs which will cause the generation of Keyboard Output Reports which will be received by USB Keyboard Receive Application.

2) USB Audio Coding Application's C# Project Code files:

Form1.cs, Form1.designer.cs and Form1.resx; USB Audio Coding Application embeds data to be exfiltrated in audio packet, creates a wave file using PCMWAVEFORMAT (Microsoft, 2014), (Stanford, CCRMA, 2003) and plays it with Windows Media Player or PlaySound() API and audio packets of the played audio will be received by USB Audio Receive Application.

The basic knowledge about C# programming with Visual Studio is sufficient to build a complete C# project from the files: Form1.cs, Form1.designer.cs and Form1.resx. The total size for both the applications is based on 3 files to be uploaded to computer system in the form of key presses is 52KB as shown in Table 2.
The time required to Upload Code was calculated. We observed that within 1 second approximately about 30 key presses can be typed using the Upload Application. Therefore the time TU seconds is the time required to upload these files of size SU in bytes and the number of key presses NK typed per second is given as

$$\text{Time (TU)} = \text{SU/NK} \quad (4)$$

Table 2 Size of Applications to Be Uploaded

Name of Application	Files	Size of each file	Size of Application
Keyboard LED Coding Application	Form1.cs	4 KB	14 KB
	Form1.disigner.cs	4 KB	
	Form1.resx	6 KB	
USB Audio Coding Application	Form1.cs	20 KB	38 KB
	Form1.disigner.cs	11 KB	
	Form1.resx	7 KB	

Total size of files to be uploaded is 38 + 14 = 52KB and the Time required to upload these files is {52*1024/30} = 1775 seconds. Also to notify the start of a new file about 5 to 10 seconds gap is used. That implies approximately the time of 30 minutes(1800 seconds) is required to upload the applications.

6.3 Results of Attack

We carried out number of attacks and made the analysis of the observed results. We attacked the computer system using four different methods as discussed below:

Method 1 (Uk): Keyboard LED Channel Attack:

We put USB Keyboard Receive interface on device and connected the device to the computer system. We started Keyboard LED Coding Application on computer system. Data to be exfiltrated is provided as an input to the Keyboard LED Coding Application. As given in equation (1), the maximum achievable throughput of exfiltration with Unintentional use of USB Keyboard Channel is 2.283 bytes/sec.

Table 3 Audio Representations for ASCII Characters

ASCII Value	Character	16-bit Audio Representation
62	>	15933
63	?	16190
64	@	16447
65	A	16704

The data was exfiltrated in the form of toggling keyboard LEDs using a VBScript file generated by Keyboard LED Coding Application. We observed that the writing speed of USB Keyboard Receive interface on Mbed LPC11U24 device is very low when it is acting as an USB Keyboard device. Most of the times, it was not able to write anything on the device. Sometimes it was able to write the data decoded from the keyboard output reports received from computer system to the local storage of the device but not more than 3 or 4 characters with insertion and deletion errors.

Only the three or four characters of exfiltrated text were stored in a file and all were from first word. Other letters were not stored in file due to the many deletion errors and slower File I/O operations on the local storage of device when it is acting as keyboard.

Method 2 (Ua with PlaySound() API): USB Audio Channel Attack With PlaySound() API:

USB Audio Receive interface is on the device and device is connected to the computer system. It recognizes itself as an USB Speaker. USB Audio Coding Application is started on the computer. It takes the input as the text file that is to be exfiltrated. First it embeds 94 ASCII Characters into the audio packet and creates two wave files and plays them using PlaySound() API; each for 5 seconds duration. Then it embeds the data from the input text file in audio packets to create and play a separate wave file for every data section of 45 bytes; each of 5 seconds duration. Also there is 2 seconds gap after playing every wave file. Now we can calculate the observed throughput for a message of MO bytes as the original message length, length of each data section is D bytes, TS is the time required to exfiltrate each data section of D bytes, TG is the time gap after playing a wave file and ME is the length of exfiltrated message in bytes.

$$\text{Throughput (bytes/second)} = \text{ME}/\{(\lceil \text{MO}/\text{D} \rceil + 2) * (\text{TS} + \text{TG})\} \quad (5)$$

We decided to exfiltrate the file of 753 bytes. The number of data sections considering the overhead of 2 data sections used for ASCII are {(MO/D) + 2} = {(753/45) + 2} = {17 + 2} = 19. The value of (TS + TG) = (5 + 2) = 7 seconds. For the experiment with this method original message length (MO) is 753 bytes and exfiltrated message length (ME) is 747 bytes. Therefore the throughput of exfiltration is {747/ (19*7)} = 5.62 bytes/sec. For a message size 96 Kbytes and above, maximum throughput of 6.42 bytes/sec can be achieved.

Table 4 Accuracy of Exfiltration

Method of Exfiltration	Percentage Word Accuracy
Method 2 (Ua with PlaySound() API)	81.3 %
Method 3 (Ua with WMP)	50.41 %

Total 753 bytes were exfiltrated to the device in the form of audio packets and device received those audio packets and stored them as 16 bit integer values to the local storage. The first two wave files were for ASCII characters, which are used to map the ASCII characters with their equivalent 16 bit audio representations. These two wave files are always exfiltrated before the data, because 16 bit audio representations may not remain same for each exfiltration, they may change to some other representations. The first two wave files are the basic requirement of exfiltration, because they are used for decoding the audio values stored on the device. These representations are mapped to equivalent ASCII characters using Microsoft Access database. Some of the representations are given in Table 3. Audio representations may take any nearby value. For example as given in Table 3, instead 16704, the character 'A' may be represented as 16701, 16702, 16703, 16705, 16706, 16707 etc. All the values are decoded to the character 'A'.

The data of 753 bytes were exfiltrated to the device in the form of separate wave files. They were stored on device in the form of their audio representations. These audio representations are decoded using USB Audio Decode Application. After decoding the text, the word accuracy was calculated using Microsoft Word Spellchecker Component and by comparing with original data. We can write the formula for calculating the Word Accuracy W,

$$\text{Percentage Word Accuracy(W)} = (\text{NC}/\text{NW}) * 100 \quad (6)$$

Where,

NC: is Number of Correct Words in the exfiltrated data.
NW: is Number of Words in the original data

We calculated the word accuracy NC = 100 and NW = 123. Therefore Percentage Word Accuracy (W) is {(NC/NW) * 100} = {(100/123) * 100} = 81.3 %. Optionally, Microsoft Word Spellchecker Suggestions can be used to apply word corrections for the incorrect words and we may get closer to the 100% accuracy. Even if we didn't succeed in achieving the 100% accuracy, 81.3% is the sufficiently good accuracy.

Method 3 (Ua with WMP): USB Audio Channel Attack with Windows Media Player:

USB Audio Receive interface is on the device. Device is connected to the computer system. It recognizes itself as an USB Speaker. USB Audio Coding Application is started on the computer with the input as the text file which is to be exfiltrated. First it embeds 94 ASCII characters into audio packet to create two wave files and plays them using WMP (Windows Media Player); each for 5 seconds duration. Then it embeds data of the input text file into audio packets to create and play a separate wave file for every data section of 45 bytes; each of 5 seconds duration. Also there is 2 seconds gap after playing every wave file.

The data of 753 bytes was divided into 17 data sections each of size 45 bytes and exfiltrated to the device in the form of separate wave files. They were stored on device in the form of their audio representations. After decoding the data, we recognized many deletion errors. This may be due to the inconsistent behavior of WMP which was also observed in (Clark, *et al.,* 2009). Using equation (5), the throughput of exfiltration is calculated for this method. The throughput value considering the overhead of 2 data sections for ASCII characters is: {592 / (19 * 7)} = {592 / 133} = {592 / 133} = 4.45 bytes/sec. We cannot guarantee the same throughput for all the exfiltrations using WMP.

Using equation (5), the word accuracy with Number of correct Words in exfiltrated data (NC) = 62 and Number of Words in original data (NW) = 123, is calculated. Therefore Percentage Word Accuracy (W) is (NC/NW) * 100 = (62/123) * 100 = 50.41 %. Optionally, Microsoft Word Spellchecker Component can be used to apply word corrections for the incorrect words and improve the accuracy. Table 4 comprises the accuracies of exfiltration for Method 2 and Method 3.

Method 4 (Ua with PlaySound() API, Store Packet): USB Audio Channel Attack With Play Sound API, Store Packet:

USB Audio Receive interface was modified so that it can be used to store the packet instead of separate integer values and the device is connected to the computer system. It recognizes itself as an USB Speaker. USB Audio Coding Application is started on the computer. It takes the input as the text file. It embeds data of the input text file repetitively into the audio packets to create and play a separate wave file using PlaySound() API; each of 5 seconds duration. Also there is a 2 seconds gap after playing every wave file.

We observed that bytes stored on the device were the plaintext characters. All the stored data was similar to exfiltrated data in plaintext. No decoding was needed. The device received 285 bytes when a single wave file of 5 seconds duration was played. Considering the gap of 2 seconds after each wave file is played, the throughput is {285/7} = 40.71 bytes/sec. In the received data, there were many insertion and deletion errors. These errors can't be corrected easily. The attacker cannot do much to correct the incorrect words. Without the knowledge of original data it is very hard to read. Percentage word accuracy is not calculated because of many errors in the data. With a good accuracy it can become the simplest way of exfiltration using Unintended USB Audio Channel.

With a high quality device Method 4 may give the better results. In our case we can conclude that Method 2 provides better results as compared to other methods discussed above. Table 5 summarizes the throughputs of exfiltration for all the methods.

Table 5 Throughput of Exfiltration

Method of Exfiltration	Throughput (Bytes/Second)
Method 1 (Uk)	2.283 bytes/sec
Method 2 (Ua with PlaySound() API)	5.62 bytes/sec
Method 3 (Ua with WMP)	4.45 bytes/sec
Method 4 (Ua with PlaySound() API, Store Packet)	40.71 bytes/sec

Uk= Unintentional use of USB Keyboard Channel; Ua= Unintentional use of USB Audio Channel; WMP= Windows Media Player.

6.4 Solution Applied

Finally, we applied the solution to test the successful prevention of Unintentional use of USB Channels. As discussed in section 5.2, the Protection Software is developed in two parts Sk and Sa. It is clear that both the solutions will work in parallel at the same time.

1) Sk- Solution to block the Unintentional use of USB Keyboard Channel:

The solution will keep looking the memory storage for VBScript files. When it sees the VBScript file is saved on the computer, it checks the file. If it is found that the file may result in Ik, i.e. the illegal activity that can be performed by making Unintentional use of USB Keyboard Channel, such file is immediately removed from the memory. If no suspicious code found in the file after scanning, the file will stay on the storage disk. More details about the solution are available in section 5.2.

2) Sa- Solution to block the Unintentional use of USB Audio Channel:

The solution will keep looking the storage disk for audio wave files. Whenever the wave file is saved on the computer, it will scan the file. If it is found that the file may result in Ia, i.e. the illegal activity that can be performed by making Unintentional use of USB Audio Channel, such file is immediately removed from the memory. If no suspicious data found in the wave file, the file will stay on the storage disk.

More details about the solution are available in section 5.2. The probable solution is implemented in this project is based on the theme of Real Time Protection. The technique used for developing the Protection Software is based on the ways that are currently available for exfiltration by making Unintentional use of USB Channels. In future, if the new ways exfiltration are found then the solution must be updated to deal with them.

Conclusion

In this paper, we examined the issues related to the Unintentional use of USB Channels; especially, the

exfiltration of data by making Unintentional use of USB Keyboard Channel and Unintentional use of USB Audio Channel. Here we first implemented the Hardware Trojan horse device based on Unintentional use of USB Channels and then we provided the solution to block or prevent them.

We have implemented Hardware Trojan horse device based on Unintentional use of USB Channels using a low cost device: Mbed LPC11U24 Microcontroller. Due to the local file storage restrictions imposed by the device and the chosen coding scheme, we got the low throughput of exfiltration. The throughput for the exfiltration by making Unintentional use of USB Keyboard Channel is 2.283 bytes/s and the same for Unintentional use of USB Audio Channel is 6.42 bytes/sec. Though the throughput of exfiltration using the developed Hardware Trojan horse device observed in this dissertation is low as compared to previously developed Hardware Trojan horse device (Clark, *et al., 2009*), then also it is good enough for the successful insider attack. This project implements a low cost Hardware Trojan horse device on the basis of Unintentional use USB Channels. Also it proves that the Unintentional use of USB Channels can be prevented using the probable solution implemented in this project. The solution is based on the concept of Real Time Protection (Microsoft, 2012). Real Time Protection means identifying the problem before it becomes threat. The implemented solution identifies the things that may cause exfiltration of data over Unintended USB Channels and blocks it before the exfiltration starts.

The implemented solution to block Unintended USB Channels cannot be the only solution, but it is one of the possible solutions to block Unintentional use of USB Channels for the exfiltration of data. Unintentional use of USB Channels can't be neglected while developing a better Endpoint Protection Software. The research about the Unintended USB Channels has been considered in the Foundational Study of Unintentional Insider Threats at Carnegie Mellon University (Bureau, Federal Infrastructure Protection, 2013). Now it's time that Endpoint Security Vendors should take a note of this.

Acknowledgment

We are thankful to our college, Pune Institute of Computer Technology, Pune, Maharashtra, India for the ideal support and patience during the research.

We are also thankful to the author J. Clark for resolving our queries about his research based on Unintended USB Channels.

References

USB Implementers Forum, (2001), USB 2.0 Specification, http://www.usb.org/developers/docs

S. Stasiukonis, (2006), Social engineering, the USB way, http://www.darkreading.com/security/article/208803634/social-engineering-the-usb-way.html

Centenial Software, (2012), DeviceWall: Endpoint Security homepage, http://www.frontrange.com/ProductsSolutions/Detail.aspx?id=9416

CheckPointSoftware, (2009), Pointsec protector homepage, http://www.checkpoint.com/products/datasecurity/protector

Device Lock Inc., (2015), "Devicelock homepage," http://www.devicelock.com

John Clark, Sylvain Leblanc, and Scott Knight, (2011), Compromise through usb-based hardware Trojan horse device, Future Generation Computer Systems, vol. 27, no. 5, pp. 555–563, doi:10.1016/j.future.2010.04.008.

John Clark, Sylvain Leblanc, and Scott Knight, (2009), Hardware Trojan horse Device Based on Unintended USB Channels, Network and System Security, NSS '09. Third International Conference on, vol., no., pp.1,8, doi: 10.1109/NSS.2009.48

D. Barral and D. Dewey, (2005), Plug and Root, the USB Key to the Kingdom," http:// www.blackhat.com/ presentations/bh-usa-05/BHnUSn05-Barrall-Dewey.pdf

M. Al-Zarouni, (2006), The reality of risks from consented use of USB devices, in Proc. Fourth Australian Information Security Conference, pp. 5–15.

G. Shah, A. Molina, and M. Blaze, (2006), Keyboards and covert channels, in Proc. the 15th conference on USENIX Security Symposium.

Shiyanovskii, Y. Wolff F., Rajendran, A.; Papachristou, C.; Weyer, D.; Clay, W.; , (2010), Process reliability based Trojans through NBTI and HCI effects, Adaptive Hardware and Systems (AHS), NASA/ESA Conference on , vol., no., pp.215-222.

Ronald W. Smith, G. Scott Knight, (2008), Predictable Design of Network-Based Covert Communication Systems, Security and Privacy, IEEE Symposium on, vol., no., pp.311-321, 18-22.

Xuehui Zhang; Tehranipoor, M.; , (2011), Case study: Detecting hardware Trojans in third-party digital IP cores, Hardware-Oriented Security and Trust (HOST), IEEE International Symposium on , vol.,no., pp.67-70.

Kangbin Yim, Soonchunhyang, (2010), A New Noise Mingling Ap-proach to Protect the Authentication Password, In International Conference on Complex, Intelligent and Software Intensive Systems (CISIS).

Kyungroul Lee; Wansoo Kim; KwangjinBae; Kang-binYim; , (2010), A Solution to Protecting USB Keyboard Data, Broadband, Wireless Computing, Communication and Applications (BWCCA), International Conference on , vol., no., pp.108-111.

PLX Technology, (2008), Net2280 home page, http://www.plxtech.com/products/net2000/net2280.asp

Mbed, (2012), Mbed NXP LPC11U24 home page, http://mbed.org/handbook/mbed-NXP-LPC11U24

Pravin Phule, (December 2012), A Low Cost Hardware Trojan horse Device Based on Unintended USB Channels, In International Journal of Advanced Computer Research, Volume2, Number 4, Issue 7, pp 114-118.

USB Implementers forum, (2001), USB Device Class Definition for Human Interface Devices (HID) 1.11, http://www.usb.org/developers/devclass_docs/HID1_11.pdf

USB Implementers forum, (2006), USB Device Class Definition for Audio Devices 2.0, http://www.usb.org/developers/devclass/docs/Audio2.0/fifin.zip.

Microsoft, Multimedia Structures, (2014), PCMWAVEFORMAT structure, http:// msdn. microsoft. com/ en-us/ library/ windows/ desktop/ dd 743663(v=vs.85).aspx.

Sourceforge projects, (2013), Teraterm Terminal Emulator, http://sourceforge. jp/projects/ttssh2/releases/

Mbed, (2013), Mbed Online Compiler, https:// mbed.org/handbook/ mbed-Compiler.

Tarasco, (2003), EXEtoVBS: VBS executable Encoder, http://www.tarasco.org/security/exe_to_vbs_encoder/index.html.

McAfee Inc., (2014) Virus Profile: Generic.df, http://home. mcafee.com/VirusInfo/ VirusProfile.aspx?key=141407

Pravin Phule, (2013), A Low Cost Approach towards Unintended USB Channels, at C-PGCON: Second Post Graduate Symposium for Computer Engineering.

Bureau, Federal Infrastructure Protection, (2013), Unintentional Insider Threats: A Foundational Study, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.366.4437&rep=rep1&type=pdf

H. Salmani, M. Tehranipoor, J. Plusquellic, (2009) New design strategy for improving hardware Trojan detection and reduction Trojan activation time, in: IEEE Intl. Wkshop. on Hardware-Oriented Security and Trust, HOST'09, pp. 66–73.

Stanford, CCRMA, (2003) WAVE PCM sound file format, https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

Microsoft, (2012), Microsoft Security Essentials Product Information page, http://windows.microsoft.com/en-US/windows/products/security-essentials/ product-information