5-4 Studies on Countermeasures for Malicious USB Devices

Tatsuya TAKEHISA, Makoto IWAMURA, and Hayato USHIMARU

USB devices that adapt USB interfaces (USB keyboards and USB mouses) are prevalent. However, a new attack has been reported that a malicious USB device rewrites USB firmware, enabling USB host attached to it to do the malicious activity out of users' intention. Research community recognizes the urgency of developing countermeasure against this kind of attack, as malicious third-party in practice makes malicious USB devices and utilize them for cyber-attacks. Against this activity, we propose in this paper a novel USB hub that detects and blocks malicious USB devices by inspecting information in USB descriptors.

1 Introduction

It has been some time since USB devices became into near-ubiquitous use. USB devices, except for a very specific use, offer the convenience of connecting devices, just by inserting it into a USB connector, which led to its widespread, and USB connectors have become common feature on many computing devices including PCs.

In recent years, however, it became known that some USB devices pose serious security threats. A method to prepare a USB device with malicious firmware installed (hereafter referred to as a "Malicious USB") was made public [1], which uses USB-fuzzing technique to override the configuration fraudulently, and exploits loopholes in the OS and device drivers to intentionally halt the system.

Also, BadUSB, a type of malicious USB, was published [2]. BadUSB is a device whose firmware is modified by a third party with malicious intention, in an attempt to intercept and falsify information, as well as triggering unexpected behavior. It has also been suggested that tampering of the USB driving firmware installed on the increasingly popular Android-based smartphones and other devices enables it to perform malicious actions similar to those of BadUSB [3]. There has even been a report that suggests shipping incidents of malicious USB devices, in which malware is incorporated during the manufacturing process[4].

In recent years, other than PCs, there are a variety of computing devices with USB connectors, such as digital multifunction machines, and they are capable of printing out the data stored on USB memory or storing scanned images on it. Self-printing terminals, often located in stores, also provide USB-connection services such as printing photographs by transferring data from a digital camera. These machines also require countermeasures against malicious USB devices.

USB devices should be designed, in the first place, so that they defy modification attempts by any third party other than the original manufacturer. It has been pointed out, however, that some USB devices allow firmware alteration if an engineer has skill above a certain level of sophistication[5]. It should be noted, however, that rewriting the USB firmware to a specific purpose requires highly sophisticated knowledge. This situation resembles the security issues of embedded software.

In this report, malicious USB devices are classified into the following four categories.

- USB device with embedded malware
  The manufacturer, with some malicious intention, embeds additional functions in the USB device, enabling it to perform maliciously.
- USB device with fraudulent descriptor
  USB configuration descriptor is overridden by a fraudulent descriptor, to make the USB device perform maliciously.
- USB device with added unauthenticated functions
  Original configuration descriptor of the USB device is intentionally modified to give it bad behavior capability.
- USB device with modified functions
  Adding function(s) to the USB device without modifying the original configuration descriptor, to act maliciously.

To cope with the threat of malicious USB devices, the authors propose a new type of USB hub capable of detect-ing and blocking a malicious USB device's attempt to connect with USB hosts (PCs, digital multifunction machines, and self-printing terminals). The proposed USB hub examines the configuration descriptor of a USB device before it is connected to the PC, and blocks the connection if the device is identified as a malicious USB device.

This report consists of the following sections: Section **2** presents background knowledge, Section **3** explains the proposed method, Section **4** discusses the results, and Section **5** gives a summary.

2 Background knowledge

This chapter provides a simple overview of USB specifications, with focus on the topics of special importance in this report: control transfer and protection of the USB device.

2.1 USB specifications and control transfer

In the USB specifications, the unit that controls a USB device, such as a PC, is called a "host," and the USB device a "target."

As shown in Fig. 1 "An example of USB system configuration," a host has a host controller that oversees USB communications that run under it. A root hub with a set of ports is arranged directly under the host controller to provide connections (tree configurations) to USB devices.

The hub is capable of cascaded connections up to 5 levels, allowing to increase the number of USB ports (up to 127 USB devices can be connected to a host controller).

Four transfer modes are defined for a USB device: control transfer, bulk transfer, interrupt transfer, and isochronous transfer.

The control transfer begins with the setting of the USB device address, followed by acquisition of device configuration information for enabling the system to use the functions installed in the device. A mode of communication, called a device request, is used to run this process. Figure 2 shows a simplified representation of USB device initialization steps performed by control transfer.

As shown in Fig. 2, the USB host uses the request transactions in control transfer mode to examine what type of USB devices is connected, and what types of functions it has.

The requests are classified into three types — standard, class-specific, and vendor-specific — and each format is shown in Fig. 3. The standard requests include GetDescriptor,

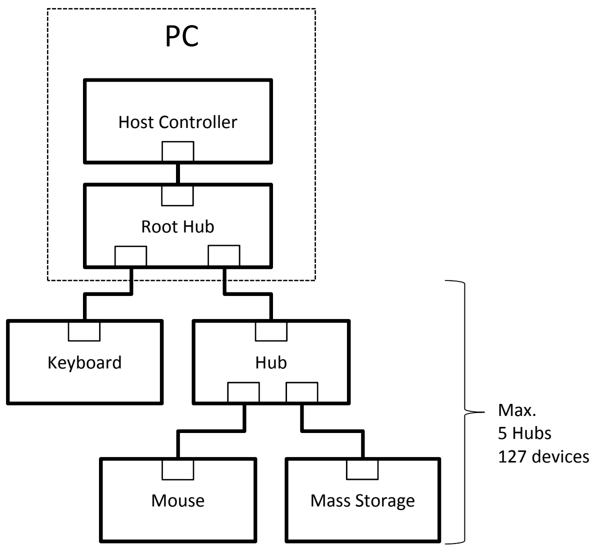


Fig. 1 An example of USB system configuration

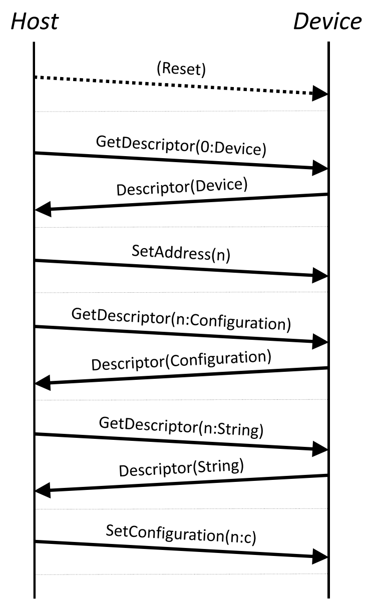
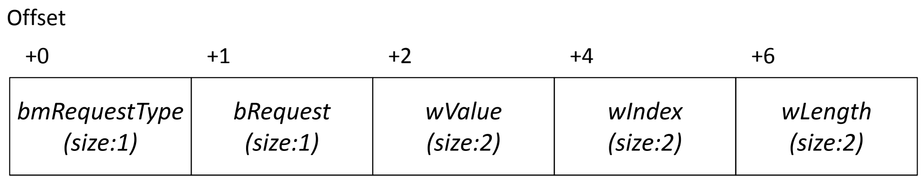


Fig. 2 An example of USB enumeration sequence


Offset

+0	+1	+2	+4	+6
bmRequestType (size:1)	*bRequest (size:1)*	*wValue (size:2)*	*wIndex (size:2)*	*wLength (size:2)*

Fig. 3 USB Device Request format

Table 1 Descriptor types (excerpts)

Descriptor Type	Value	Length
DEVICE	1	18
CONFIGURATION	2	9
STRING	3	2+N
INTERFACE	4	9
ENDPOINT	5	3

Table 2 Transmission speed for each USB standard

Version	Mode	Max. Transfer Rate

USB 1.0	LS(LowSpeed)
	FS(FullSpeed)
		1.5 Mbps
		12 Mbps

USB 1.1	“”	“”
USB 2.0	HS(HighSpeed)	480 Mbps
USB 3.0	SS(SuperSpeed)	5 Gbps
USB 3.1	SSP(SuperSpeedPlus)	10 Gbps

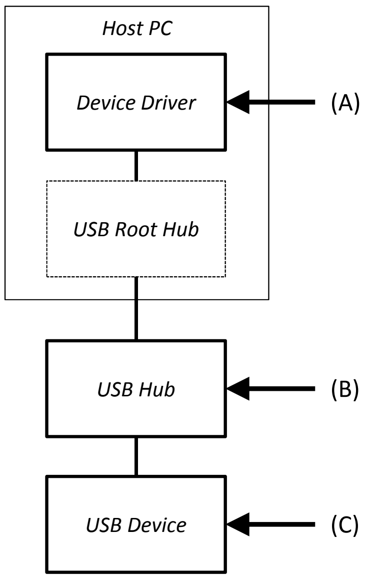


Fig. 4 Locations to implement the functions to inspect a USB device

SetAddress and SetConfiguration as described below.

- GetDescriptor request
  GetDescriptor demands the descriptors that define the device, configuration and strings. Each descriptor type has a value as listed in Table 1. The USB device, upon receiving a GetDescriptor request, sends back an appropriate descriptor that matches the requested type. Each request is defined using one of the descriptor types, and the descriptor to be returned is determined according the “length” values listed in Table 1.
- SetAddress request
  SetAddress sets an address on the USB bus.
- SetConfiguration request
  This request enables the system to select one of the functions installed on the USB device. In the case where a USB device has multiple of functions, the USB host can use this request to select one of them for optimizing system performance.

In line with a higher transmission rate and increased variety of functions, the USB specifications have undergone stepwise upgrades. Table 2 shows the list of transmission rates corresponding to each USB standard. Upward compatibility is guaranteed from USB 1.1 up to USB 3.1. Therefore, when a USB device is connected, the host first attempts to perform control transfer at the communication velocity rated for USB 1.1 on the ground of compatibility.

2.2 Protection of USB devices

There are several possible locations, as shown in Fig. 4, to implement inspection functions to protect the system from bad USBs.

In the configuration shown in Fig. 4(A), the functions provided by the OS and device drivers protect the USB host from bad USBs. This approach, however, may not be effective against the types of attacks (see, for example, Bouyat et.al [1]) that take advantage of vulnerabilities in the device driver.

In the configuration shown in Fig. 4(B), the USB hub can identify if the USB device is bad at the very moment it is connected. Thus, the bad USB cannot take advantage of vulnerabilities in the USB host. Configuring a transparent and bidirectional mediator device in between the USB host and USB device — a method proposed by Tonder et. al[6] — may enable the mediator device to block fuzzing attack against USB. This approach, however, necessarily involves reduction in communication speed. In addition, the transparent and bidirectional mediator device needs many components, increasing the cost.

In the configuration shown in Fig. 4(C), the USB device must provide certain detection logic that necessitates a high-performance device implemented in it (such as a dedicated IC, or CPU with additional functions).

Implementing the inspection capability in location (A) or (C) requires, from the considerations given above, new development of detection logic and vulnerability reduction efforts for each major components — i.e. OS, device driver, and USB device.

Placing the inspection capability in (B), however, eliminates the need to consider the problems associated with (A) and (C). Inspection performed in the USB hub at location (B) can be quite an effective option.

In this report, the authors propose a bad USB device counterapproach that takes full advantage of the locational characteristics of (B), with a minimum performance penalty in the USB device.

3 The proposed approach

In this section, the configuration and functions of the USB hub — characterized by its implementation of inspection functions — proposed by the authors (hereinafter referred to as the "proposed USB hub") are described in **3.1**, and the specific inspection method enabled by using it is explained in **3.2**.

This proposed method attempts to detect and inspect a malicious USB device by solely using control transfer defined in the USB protocol. As described in Section **2**, the control transfer that can be used for identifying the device configuration and functions of a USB device is only defined in the USB 1.1 standard. Upward compatibility of the USB standards, however, makes it possible to construct a low-cost USB hub based on the earlier standard, without the need for the higher transmission rates of USB2.0 and 3.x.

3.1 USB hub with inspection function

The proposed USB hub consists of the following elements (see Fig. 5): a USB host (an embedded CPU) for performing inspection, a USB hub (USB Hub A) for connecting to the USB host, another USB hub (USB Hub B) for connecting to the upper layer host (PC), USB switch ICs (#A - #D) for each port for connecting USB devices, and corresponding push buttons (#A - #D).

The following three modes of operation are available to the proposed USB hub.

- Registration of a USB device:
  A USB device is registered when it is inserted into a USB port while a push button with the corresponding number (#n) is pressed.
- Deletion of a USB device:
  A USB device is deregistered when it is removed from a USB port while a push button with the corresponding number (#n) is pressed.
- Recognition of a USB device:
  A USB device is recognized when inserted into a port without pressing a button.

When a USB device is registered/recognized, the inspector USB device controls the USB switch of the port, and connects it to the USB Hub A.

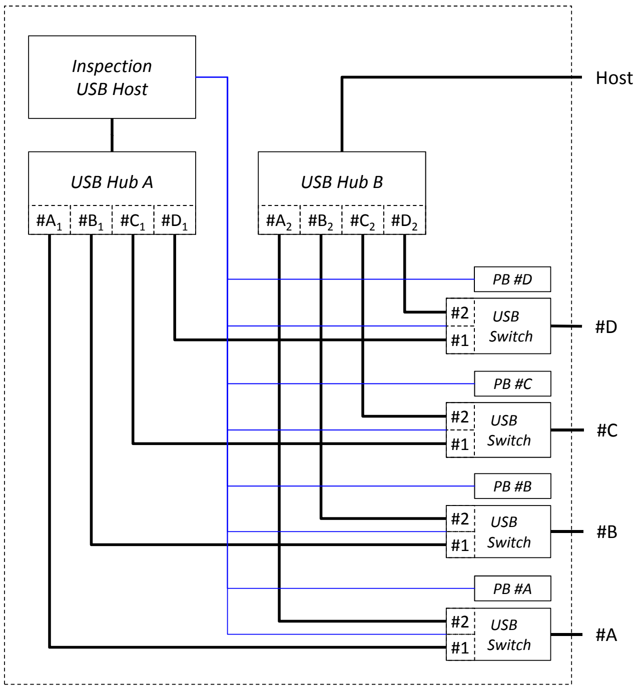


Fig. 5 Block diagram of a USB hub with inspection functions

Upon completion of the connection, the USB Hub A notifies the successful connection of the device to the inspector USB host, triggering the host to start inspection procedures.

When the inspection procedures complete successfully, the inspector USB host registers the hash value of the USB device descriptor as identification information for the device, whereby a pair of values (VID and PID) obtained during the registration procedures are used as the key.

When the inspection procedures complete successfully, the inspection USB host controls the USB switches of the USB port to connect it to USB Hub B.

In this way, the proposed USB hub performs inspection procedures before the USB device is connected to a host. If the USB device failed inspection, its functions are put on hold.

When a USB device is unregistered, the inspector USB host deletes the device's descriptor information (i.e. VID and PID).

Deletion of the information blocks the USB device when it tries to connect again. It must go through the registration procedures over again to seek a connection to the host. In the next section, the inspection procedures performed by the proposed USB hub are described.

3.2 Inspection method

In this section, the procedures taken by the inspector USB host are described.

The inspection methods are based either on a whitelist, blacklist or fraudulent descriptor, each of which requires acquisition and examination of a USB device descriptor.

The following sections provide detailed descriptions of each method.

3.2.1 Whitelist screening

Figure 6 shows the examination flow of whitelist screening.

(A) The GetDescriptor request is issued to acquire the descriptors needed (device, configuration, string, interface and end point). For each descriptor, a hash function is used to create information for device confirmation (the hash function maps the input onto a certain range of values, and it must always return the same value for each specific input. To ensure this property, use of a cryptographic hash function, such as SHA-1, is recommended wherever possible).

(B) The PID and VID are extracted from the device descriptor obtained in step (A), and, by using them, registered device information is identified in the list.

(C) The device information for confirmation is compared with the registered device information. The checking reports good (OK) if all items in both device information exactly match. Otherwise, the checking reports fail (NG).

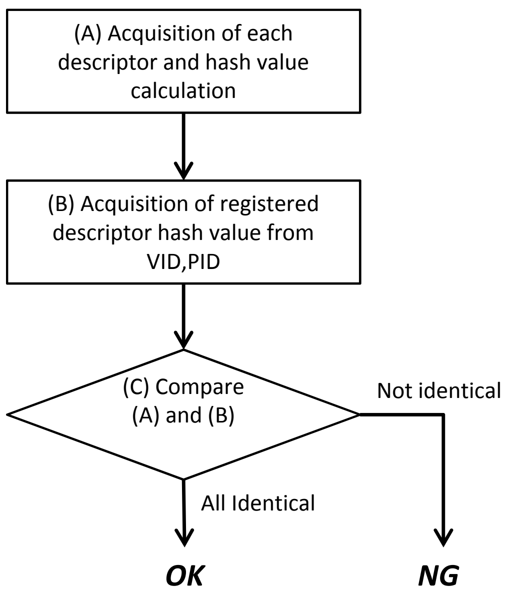

(A) Acquisition of each descriptor and hash value calculation
(B) Acquisition of registered descriptor hash value from VID,PID
(C) Compare (A) and (B)
Not identical
All Identical
OK
NG

Fig. 6 Examination flow: whitelist screening

This enables the system to judge the USB device to be identical with the registered one in terms of device, configuration, string, interface and end point descriptors, and distinguish it from a bad USB device, even if it has the same PID and VID (a bad USB device needs descriptor modification to add one or more functions to perform its malicious intentions).

USB devices of the same type can be clearly distinguished if the serial number is given by string descriptor. Therefore, only the registered one can gain an “OK” in the examination (an identical USB device fails the examination if it has a different serial number).

3.2.2 Blacklist screening

Figure 7 shows the examination flow of blacklist screening.

(A) Acquisition of the device descriptor using a GetDescriptor request.

(B) Screening of the VID and PID contained in the device descriptor against the blacklist.

The examination fails (NG) if the VID and PID coincide with those listed in the blacklist. Otherwise, the examination gives an OK.

Note that the same screening scheme can utilize other identifiers other than VID and PID — e.g. an appropriate string defined in the string descriptor, and combination of them.

This is an effective method to judge if a USB device is identical with those already acknowledged as bad.

Because of its relatively easy implementation, blacklist screening provides an effective method for rough checking.

3.2.3 Fraudulent descriptor examination

Figure 8 shows the flow of fraudulent descriptor examination.

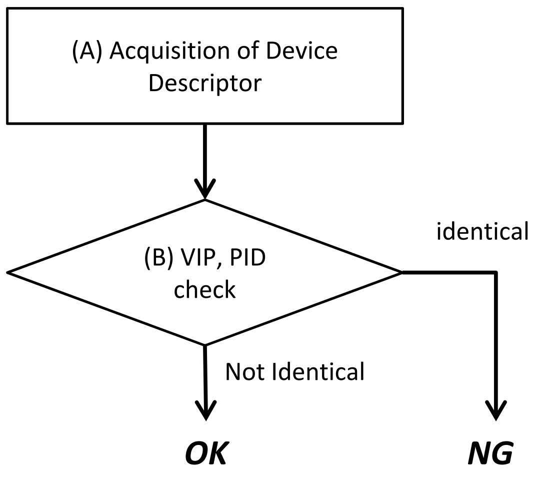


Fig. 7 Examination flow: blacklist screening

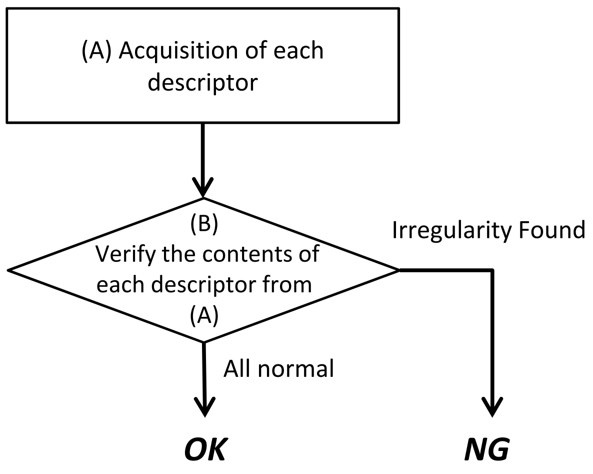

(A) Acquisition of each descriptor
(B) Verify the contents of each descriptor from (A)
Irregularity Found
All normal
OK
NG

Fig. 8 Flow of examination: fraudulent descriptor

Table 3 Examinations performed for each operation mode

Operation mode	Check item
Registration of a USB device	Blacklist screening Fraudulent descriptor examination
Authentication of a USB device	Blacklist screening Whitelist screening
Deletion of a USB device:	None

(A) A GetDescriptor request is issued to acquire the descriptors (device, configuration, string and interface descriptor). On an as-needed basis, class information can also be obtained [7] from bFunctionClass, bFunctionSubClass and bFunctionProtocol defined in the interface descriptor [8].

(B) Each acquired descriptor is checked for any fraudulent values. The items to be checked include such entries as bNumEndpoints in the interface descriptor (see Bouvat [1]), which, if set to zero, may cause a BSOD (Blue Screen of Death) on a Windows system.

3.2.4 Examination for each operation mode

This section describes the contents of examination performed by the proposed USB hub in each operation mode.

Table 3 summarizes the contents of examination in each operation mode.

As shown in Table 3, the proposed USB hub performs blacklist screening and fraudulent descriptor examination in the registration phase of a USB device. The procedures enable the system to block known bad USB devices and those with fraudulent descriptors from connecting with the USB host.

In the authentication phase of a USB device, the proposed USB hub performs blacklist screening and whitelist screening. The blacklist screening enables the system to disconnect one or more of the USB devices that were found to be bad after they passed the checks in the registration phase.

No additional examination is needed in the deletion stage.

As described above, performing appropriate examinations in each stage enables the proposed USB hub to detect bad USB devices, and block them before being connected to the USB host.

4 Discussion

Even if an attempt with malicious intentions is being made to rewrite the firmware on a USB device connected to a host, the proposed USB hub by itself is not capable of blocking the accesses (direction (A) shown in Fig. 9). This deficiency arises basically from the fact that the method of rewriting the firmware on a USB device depends on the specifications of embedded ICs. Because of the wide variety of such ICs in the marketplace, setting up a strategy on an IC-by-IC basis becomes very difficult, and some of them do not even disclose their specifications. However, placing the proposed USB hub in between the host and the USB device can change this situation. This enables the proposed USB hub to check the descriptor on the USB device in several ways (whitelist and blacklist screening, tampered descriptor detection), leading to the detection and blocking of malicious USB devices (direction (B) shown in Fig. 9). In the following part of this section, further description is provided for each type of malicious USB device classified in Section **1**.

- USB device with embedded malware
  For the proposed USB hub, blacklist screening is the main tool to detect and block this type of malicious USB device. As in the case of anti-virus software, blacklist information requires to be updated after each identification of a malicious USB device.
- USB device with fraudulent descriptor
  The proposed USB hub performs fraudulent descriptor inspection to detect and block a malicious USB device. This approach is effective against USB fuzzing and other related attacks.
- USB device with added unauthenticated functions
  The proposed USB hub performs whitelist screening to detect and block this type of malicious USB device. Whitelist screening is especially effective for detect-

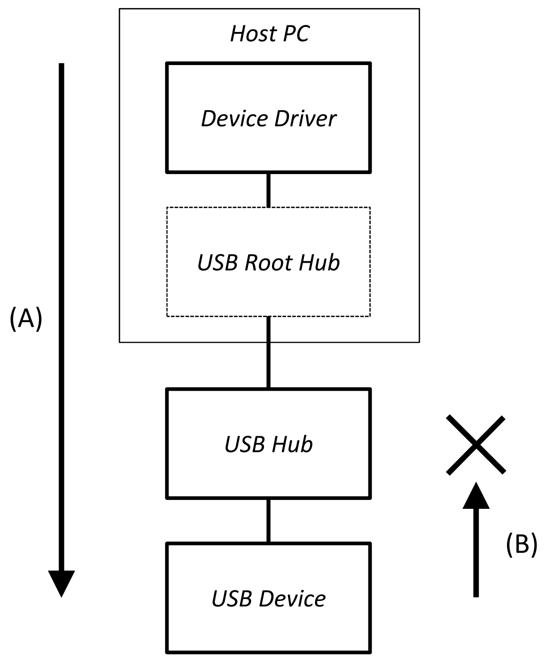


Fig. 9 Directions of inspection flow available to the proposed USB hub

ing USB devices with augmented functionalities (e.g. USB memory with an added USB keyboard).

- USB device with modified functions

  A USB device, if its functions have been modified, poses a detection problem for the proposed USB hub, because the descriptor of the USB device may remain the same even if its functions have been intentionally modified to work maliciously. For example, a very shrewd modification of firmware enables a malicious USB keyboard — designed to send fraudulent keystrokes at given intervals (while the user is away from the PC), or while no key is pressed for a certain period of time — behave perfectly innocently under normal conditions using the same, intact descriptor. The proposed USB hub cannot detect such malicious USB devices. Functional enhancement of the proposed USB hub, however, may overcome the problem. For example, a display device, such as an LCD, additionally configured into the proposed USB hub enables the system to display a one-time password immediately after a USB keyboard is connected to the hub, and demand the user to enter the same password within a specified time. Non-compliance to this demand can lead to an NG judgement. This scheme is especially effective if the malicious USB keyboard is designed to perform unintended behavior immediately after being connected to a USB connector.

In addition to the examination of descriptor used in the proposed USB hub, other methods are also conceived such as Deep Inspection used in IDS/IPS devices to monitor the USB transmission signal. However, the very high rate of data transmissions on USB 3.1 devices (typically 10 Gbps. See Table 2) makes the Deep Inspection highly resource and technology dependent. Such an application can be costly if it is ever placed on the market.

5 Summary

In this report, the authors proposed a new type of USB hub that features provisions against malicious USB devices. It provides inspection functions to detect and block any connection attempts from malicious USB devices to a USB host (e.g. PC).

To cope with the future proliferation of attacks by means of malicious USB devices, the proposed USB hub can provide an effective measure to defeat their activities.

Our challenges for the future include: circuit design and fabrication of an operational USB hub that actually features our proposals, and development of the inspection method capable of blocking such malicious USB devices that even pass through the gates of whitelist screening and fraudulent descriptor screening.

Acknowledgments

The authors wish to express their deep appreciation to Mr. Nobuyuki Kanaya (Cyber Tactics Laboratory) for the invaluable advice he gave us during the course of writing this report.

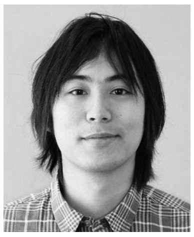
References

1 J. Bouyat, “USB Fuzzing Basics: From fuzzing to bug reporting,” Quarkslab’s blog, http://blog.quarkslab.com/usb-fuzzing-basics-from-fuzzing-to-bug-reporting.html
2 K. Nohl, J. Lell, “BadUSB -- On accessories that turn evil,” https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf, 2014.
3 BadAndroid-v0.1, https://srlabs.de/blog/wp-content/uploads/2014/07/BadAndroid-v0.1.zip
4 BadUSB Exposure, SRLabs Open Source Projects, https://opensource.srlabs.de/projects/badusb
5 “Now e-cigarettes can give you malware,” Guardian News and Media Limited, http://www.theguardian.com/technology/2014/nov/21/e-cigarettes-malware-computers
6 V. Tonder, Rijnard, and Herman Engelbrecht “Lowering the USB fuzzing barrier by transparent two-way emulation,” Proceedings of the 8th USENIX conference on Offensive Technologies. USENIX Association, 2014.
7 Usb.org, USB Class Codes, August 11, 2014, http://www.usb.org/developers/defined_class

8 Universal Serial Bus 3.1 Specification, Revision 1.0, July 26, 2013, http://www.usb.org

Tatsuya TAKEHISA
Invited Advisor, Cybersecurity Laboratory, Cybersecurity Research Institute
Cyber Security, Embedded Device Security

Makoto IWAMURA, Dr. Eng.
Former: Cooperative Visiting Researcher, Cyber Tactics Laboratory, Cybersecurity Research Center
Malware Analysis



Hayato USHIMARU
Technical Research Expert, Cyber Tactics Laboratory, Cybersecurity Research Center
Malware Analysis