UDC 004.326.1
DOI: 10.15587/1729-4061.2022.269031

DEVISING AN APPROACH TO ANALYZE THE PARAMETERS FOR DETERMINING POTENTIAL PRE-MODIFIED FIRMWARE OF USB DEVICES

Yekaterina Zuyeva
Corresponding author
Master of Applied Mathematics, Senior Teacher
Department of Information Systems and Cybersecurity
Almaty University of Power Engineering and Telecommunications named after Gumarbek Daukeyev
Baitursynov ave., 126/1, Almaty, Republic of Kazakhstan, 050013
E-mail: poka23@mail.ru

Anna Pyrkova
Candidate of Physical and Mathematical Sciences, Professor
Department of Computer Science*
*Al-Farabi Kazakh National University
Al-Farabi ave., 71, Almaty, Republic of Kazakhstan, 050040

Abdizhapar Saparbayev
Doctor of Economics, Professor
Department of International Relations and World Economy*

Aiymzhan Makulova
Doctor of Economics, Professor
School of Digital Technologies
Narxoz University
Jandosova str., 55, Almaty, Republic of Kazakhstan, 050035

Gulzinat Ordabayeva
Senior Teacher
Department of Information Systems*

This paper reports the results of experiments and studies involving different types of devices that can implement a BadUSB scenario, for example, BadUSB, Rubber Ducky, which, when connected to a computer, impersonate a device with a Human Interface Device, emulating other devices such as a keyboard and mouse.

Given the problem of the lack of management tools for detecting preliminary modifications of USB devices against attacks based on the seizure of computer control, a software and hardware system is proposed as an object of study. It is implemented programmatically in the Arduino IDE environment, and physically it is made on the Arduino Mega board with Shield, which reads the parameters of the devices. It monitors the startup of USB devices and checks each device for pre-retrofitting by passing HID descriptors from the connected hardware. Having parsed the data using Python, the data are represented in the appropriate form for analysis, on the basis of which a decision is made by the system on the possible preliminary modification of the USB drive from which these data came. This is due to the detailed consideration and thorough analysis of data, data types, temporal characteristics of data transmitted along different channels. The technical characteristics and functionality of USB devices were investigated; the parameters transmitted at the moment when they are supplied with power were determined. The system can draw a conclusion based on the analysis according to its algorithm and block a suspicious USB device that has been connected and that can intercept control over the computer.

The results of the study could be used in the field of protection of information systems from attacks based on the seizure of control from external media. The designed solution increases the level of security of the system, making it possible to recognize a possibly pre-modified device at the connection stage

Keywords: information protection, USB devices, HID, BadUSB, USB controllers, modification of USB devices

Received date 06.10.2022
Accepted date 16.12.2022
Published date 30.12.2022

How to Cite: *Zuyeva, Y., Pyrkova, A., Saparbayev, A., Makulova, A., Ordabayeva, G. (2022). Devising an approach to analyze the parameters for determining potential pre-modified firmware of USB devices. Eastern-European Journal of Enterprise Technologies, 6 (9 (120)), 51–58. doi: https://doi.org/10.15587/1729-4061.2022.269031*

1. Introduction

Companies face attacks based on vulnerabilities in USB devices that reprogram existing devices so that they are perceived by the system as another device. Embedded malicious code affects the security and stability, correct operation of computer systems and organizations, turning them into distributors of malicious code and as a result, economic, technical, production and reputational losses occur. This vulnerability is not saved from by any antiviruses, or any other solutions - they are not able to give an adequate assessment of the actions of information carriers.

The widespread use of USB devices for data storage and transmission is due to their versatility, reliability, performance, simplicity, and convenience. At the same time, USB devices are among the most dangerous and actively used means and channels for the implementation of information security threats. There is a problem of the lack of tools to control the detection of preliminary modifications of USB devices.

Therefore, research on the development of an analysis of a possible preliminary modification of USB devices is relevant.

2. Literature review and problem statement

Every interaction between USB devices and the computer is carried out using a microcontroller and so that it can carry out operations, control code is stored in its own service memory, which is triggered as soon as power is supplied to the device. With this block of memory, the user under a special mode can work and program it. Most defense mechanisms come down to the user's participation in making a decision about trust and distrust of a particular device. There is an approach to detecting suspicious USB devices by analyzing the temporal characteristics of the traffic of the USB packets they generate. This approach is similar to intrusion detection methods in network systems [1]. The software made for the Linux operating system and the approach are interesting because it does not involve the user in the decision to trust some USB device. The experimental implementation of the concept for Linux assesses the effectiveness and efficiency of the approach to cope with temporary changes in the characteristics and dynamics of the set of characteristics of legitimate users. The types of devices that can implement the BadUSB-scenario are considered. These are BadUSB, IRON-HID, Rubber Ducky, BadAndroid, BadBIOS and hacking devices based on the tiny Teensy dev board (USBdriveby and Teensyterpreter). When connected to a computer, they impersonate a Human Interface Device, emulating a keyboard and mouse. When formulating threat models, they assume that all USB packets are correctly formed and valid according to the USB protocol. No experiments have been conducted in the Windows operating system. The concept did not accept USB attacks related to the autorun feature that exploit vulnerabilities in the USB device drivers of the host operating system caused by malformed USB packets but this was noted in work [2]. Execution timings from different devices (interrupts) were detailed and measured, where the speed of keystrokes is analyzed, tracking the state of the operating system with further integration into its decision-making logic to block malicious USB traffic. In addition, the software USBlock developed in [1] cannot protect against attacks launched at the BIOS level, as is the case with BadBIOS.

A serious problem may be the attempt to improve the organizational security of USB devices within the framework of a security model separately removed from the system, as was implemented in [3]. In trust management schemes, the TMSUI software, acting as a firewall, organizes the security system, make it possible to connect USB drives only on separate terminals for a certain time. When analyzing the 6 security properties, TMSUI decides only about temporary access only to certain secure terminals.

A common type of BADUSB attack often occurs by running a Powershell script using BadUSB, which activates the Keylogger [4]. The script is written in the Arduino IDE and runs on the Pro Micro microprocessor board, which makes it possible to produce multiple attacks and fix the execution time on different architectures. But the time required to run the keylogger is 7.474 seconds and when the computer is connected to the Internet, the results of the malicious script are sent to the attacker by e-mail. The authors note that there are no studies to correct the delay in script execution.

Work [5] proposes a strategy for protecting against USB attacks - non-interference of the user. It is concluded on the basis of an analysis of what is happening in the user's session, the possibility of the origin of events in the operating system is calculated, for example, if the machine is inactive, then at this time the connected active devices are recognized as illegitimate. The proposed approach analyzes the origin of keystrokes when making automated decisions on security; and in cases where the architecture of the network configuration changes. It all comes down to determining who gives the commands – the user himself or the malicious component. Two attack vectors via USB and vectors of corresponding protection are written about. In the first case, the use of drivers (protection is reduced to the use of secure encoding and amortization of the consequences of execution errors). The second is user emulation (protection does not make it possible to bypass errors in the code but masks the device as another). The key point of protecting the proposed approach is the use of masks, filtering keys that are considered to be dangerous. The approach is implemented for both the Linux and Windows operating systems. If one implements long delays in the sketch, then the proposed system can be circumvented.

Paper [6] considered Spyduino – a board programmable using the Arduino IDE and recognized in the operating system as a HID device with great potential. On its basis, attackers can implement scripts, in turn, embedding it in any devices that will further create a BadUSB vulnerability. The board is built into the USB keyboard and sends confidential information to the FTP server without the user's permission, because Spyduino is an Arduino Uno with special libraries (expanding its capabilities) that can significantly damage everyone and everything. The article discusses various countermeasures and presents possible extensions but the policy of applying whitelists is not always applicable since the device may be clean at one point in time, and in another case modified, and since it is already on the whitelist, the system will be attacked by this USB device.

Since BadUSB modifies the USB firmware and can attack all systems to which an infected USB is connected, a good approach is considered to be the applicable solution against this generation of malicious software - to apply a whitelist [7]. A detailed approach to fingerprinting based on USB functions is proposed, which facilitates the creation of a list of trusted USB devices. The solution prevents and detects BadUSB and similar attacks, generating fingerprints based on the functions of trusted USB devices and their primary use. The information was tested for the uniqueness of the fingerprints generated by the authors. The analysis data was collected from USB drives of academic laboratories for 1 year based on functions with a whitelist participant recognition accuracy of 98.5 %. According to statistics, 7–13 % of all erroneous results are explained by equipment malfunctions or incorrect identification of samples in the laboratory. Therefore, laboratory data obtained in [7] objectively have an error.

There are cases when it is proposed to pre-filter data from USB devices [8] and the vast majority of these filters are implemented at the operating system level, leaving one of its parts and the firmware of the USB host controllers unprotected. Therefore, the authors offer flexible, universal USB filtering policies. They explore the differences between

USB data exchange filtering at the level of operating system and at the USB packet transfer level. This could be used in the early stages of device recognition and security assessment of packet filtering policies.

Study [9] shows the work of the software Firm USB ware analysis framework, which uses scripts and knowledge of the USB protocol to determine and correlate the images of embedded programs on the device and determine the activity that they can produce. FirmUSB checks the firmware of the device to determine its ability to generate potentially malicious behavior. Determining whether a device is secure is challenging because of the many different device architectures and operating systems. The authors performed experiments to measure the time of obtaining target instructions related to USB for only two firmware Phison 2251-03 and Cypress EZ-USB. And now these are not the most popular controllers.

The optimal solution should combine a mixture of several strategies [1–9], and not only meet the requirements in any one approach, as described above, but also meet the requirements in the field of safety. With the increase in the number of USB devices implemented on different controllers, vulnerable points in the system, the ability of intruders to hack, steal, and compromise information increases, it means that it is necessary to create a software package for determining the parameters of transmission from USB devices, implemented in hardware as part of an external board with a separate microcontroller.

A detailed, expanded analysis of the parameters of USB devices when they are connected is needed, as well as a tool for deciding on the possible pre-modification of devices through the analysis of descriptors and data structures issued by them. In this method, it is necessary to analyze the parameters issued by the devices at all stages of the survey by the operating environment and detail in justifying the conclusion.

3. The aim and objectives of the study

The purpose of this study is to develop an approach to the analysis of the parameters for determining the possible pre-modified firmware of USB devices. This will create an adequate tool for making a decision on the pre-modification of USB devices based on high-quality data analysis.

To accomplish the aim, the following tasks have been set:

– to obtain descriptors from USB devices of the test site;

– to synthesize the structure of the representation of data descriptors from devices;

– to parse the modes of operation of USB-devices.

4. The study materials and methods

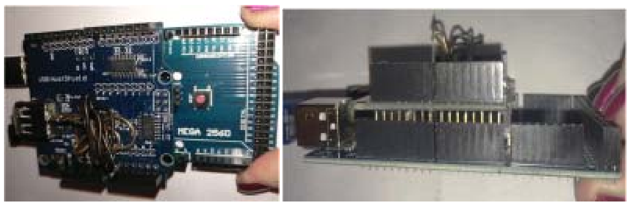
The object of our study is the transmitted characteristics of USB devices. During all the studies, about 40 devices were analyzed.

Research methods: the technical characteristics and functionality of USB devices were investigated, the parameters transmitted to the system when power is supplied were determined.

The hypothesis of the study is to implement a system on an external board in the software and hardware environment in order to isolate the USB drive from the operating environment in order to be able to listen to the temporary transmitted characteristics of the devices, the detailed parameters of the description of the devices. At the same time, we believe that some intermediate environment is created, and all devices are connected to it. Further measurements are carried out precisely inside the software and hardware system.

Experiments and studies have been conducted with different types of devices that can implement BadUSB scenarios: BadUSB, Rubber Ducky. These were pre-modified devices. And when they turned on the power, connecting to the system, they pretended to be various other devices, for example, a keyboard and a mouse.

To be able to read the incoming characteristics of devices and respond, a software and hardware system was implemented. The software was created using the Arduino IDE. Since the system is physically implemented on the Arduino Mega board with Shield, which reads the parameters of the devices, it is possible to track and transfer HID descriptors from the connected equipment (Fig. 1).



a *b*

Fig. 1. Arduino Mega board with Shield card reader to protect against BadUSB vulnerability: *a* – top view; *b* – side view

The connection to the computer of the board is made using a separate adapter UART-USB. A device is inserted into the USB port to check if it is pre-modified. The program is written in the Arduino IDE in the C/C++ programming language with the inclusion of many libraries, including the USB host libraries. In the process of work, a port is created (emulated) and descriptors of the connected device are read through it. Having parsed the data using Python, it is represented in the necessary form for analysis. The program checks the data of the descriptors from the interface and device sections (that is, it looks for changes and unnecessary descriptions in the descriptors, primarily in the product identifier, the manufacturer ID, and the device class code), on the basis of which a conclusion is made about a possible preliminary modification of the USB drive from which this data came.

We investigated the technical characteristics and functionality of USB devices, determined the parameters transmitted to the system when power is supplied, and the port is emulated; the identifiers of the pre-modified devices differed in most cases from the original ones by one character or it was impossible to identify or find a match.

To compare the original devices with the modified ones, several multi-chip USB devices were purchased with their further flashing in the Duckuino language. After a preliminary modification, these devices began to broadcast, creating a WI-FI channel. In the course of experiments, it turned out that these devices created a channel in both Linux and Windows with normal, stable broadcast quality. Further, the stitched sketches were launched through an organized channel either using the button control menu or by launching sketches [10–12].

Not multifunction devices (e. g., MFD: scanner-copier-printer) but devices that perform only 1 function should have a single interface descriptor. For example, in one experiment, a situation arose when in a single-function modified USB device the interface descriptor (Fig. 2, *a*) was represented by two interface descriptors (Intf number 0 and Intf number 1) simultaneously.

When the operating system identifies a device as the type of USB device to be connected, the system was given a descriptor that had an Intf number of 1 and another descriptor with a value of 0 was not read. In addition, the Intf Class interface descriptor had a value of 3, which means that the class of this device is a HID device. In fact, the Intf Class device has a value of 8, and this is Mass Storage. The factory settings of USB devices have the correct values in the Intf Class field (Fig. 2, *b*).

Thus, all devices were checked. With the help of the developed software and hardware system, all flashed USB devices were detected. A system of pre-modified USB devices identified everything and issued an appropriate warning. This fact means that the created system detects the BadUSB vulnerability in USB devices. So, its use will increase security for both production stations and personal computers.

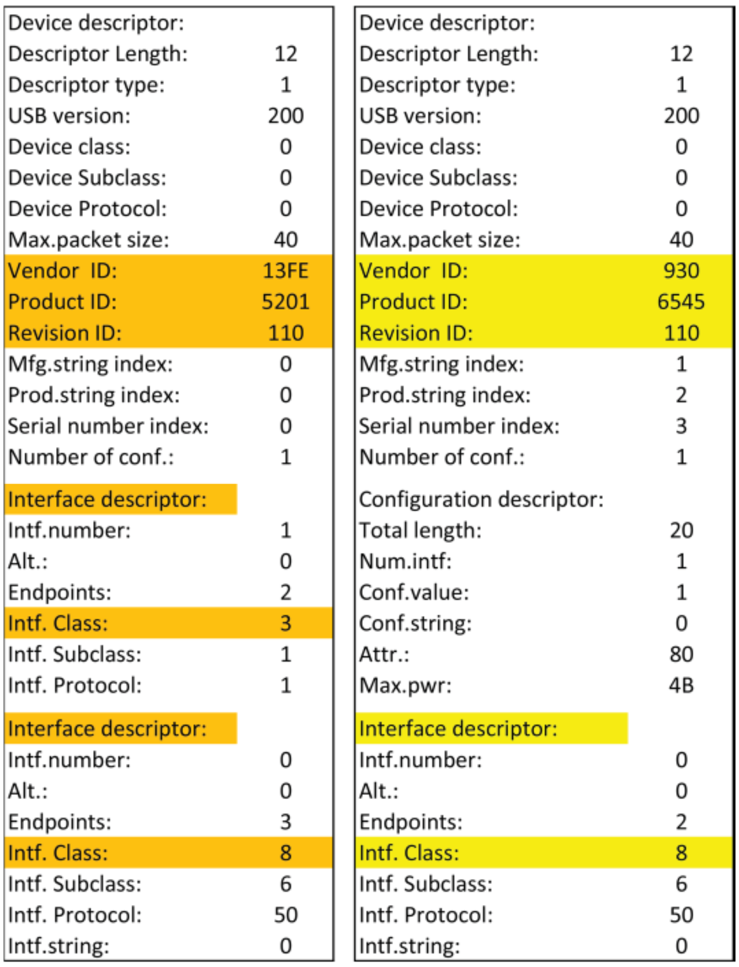
5. Results of data acquisition and analysis

5. 1. Retrieving descriptors from test polygon devices

In the task of recognizing which device is physically inserted into the computer, the USB is designed in such a way that the host controller recognizes the action of inserting and removing the connector plug. When a plug event occurs, the host controller informs its device driver, which scans the bus and asks the device to identify itself. USB devices contain descriptors, a set of information about the device.

Device descriptors are retrieved from all devices using a sketch written in the Arduino IDE (Fig. 3).

The data at this point allow the device driver for the USB bus itself to efficiently request a new connected device and wait for a response. As a test site for this study, a total of 12 devices with different characteristics were taken from 40 devices: iPhone, joystick, keyboard, mouse, game mouse, LilyPad Arduino, USBmodem, web-camera, USB 3G modem, Android-device, doc-station, Mega Arduino. Part of these 8 devices: iPhone, joystick, keyboard, mouse, game mouse, LilyPad Arduino, USBmodem, web-camera are shown in Fig. 3. In the first step, some of the descriptors are used. The algorithm defines device classes to connect a system driver for each class (which is sufficient to treat any devices in that class).



Device descriptor:	
Descriptor Length:	12
Descriptor type:	1
USB version:	200
Device class:	0
Device Subclass:	0
Device Protocol:	0
Max.packet size:	40
Vendor ID:	13FE
Product ID:	5201
Revision ID:	110
Mfg.string index:	0
Prod.string index:	0
Serial number index:	0
Number of conf.:	1
Interface descriptor:	
Intf.number:	1
Alt.:	0
Endpoints:	2
Intf. Class:	3
Intf. Subclass:	1
Intf. Protocol:	1
Interface descriptor:	
Intf.number:	0
Alt.:	0
Endpoints:	3
Intf. Class:	8
Intf. Subclass:	6
Intf. Protocol:	50
Intf.string:	0

a

Device descriptor:	
Descriptor Length:	12
Descriptor type:	1
USB version:	200
Device class:	0
Device Subclass:	0
Device Protocol:	0
Max.packet size:	40
Vendor ID:	930
Product ID:	6545
Revision ID:	110
Mfg.string index:	1
Prod.string index:	2
Serial number index:	3
Number of conf.:	1
Configuration descriptor:	
Total length:	20
Num.intf:	1
Conf.value:	1
Conf.string:	0
Attr.:	80
Max.pwr:	4B
Interface descriptor:	
Intf.number:	0
Alt.:	0
Endpoints:	2
Intf. Class:	8
Intf. Subclass:	6
Intf. Protocol:	50
Intf.string:	0

b

Fig. 2. Devices settings: *a* – modified; *b* – original settings

Parametrs	iphone	joystick	keyboard	mouse	game mouse	LilyPad Arduino	USB WIFI modem	web camera
Device descriptor:								
Descriptor Length:	12	12	12	12	12	12	12	12
Descriptor type:	1	1	1	1	1	1	1	1
USB version:	200	110	110	110	110	200	200	110
Device class:	0	0	0	0	0	EF	0	FF
Device Subclass:	0	0	0	0	0	2	0	0
Device Protocol:	0	0	0	0	0	1	0	0
Max.packet size:	40	8	8	8	8	40	40	8
Vendor ID:	05AC	1345	046D	093A	09DA	1B4F	148F	0AC8
Product ID:	1297	1000	C24B	2510	7479	9208	5572	303B
Revision ID:	310	100	7320	100	9900	100	101	100
Mfg.string index:	1	0	1	1	1	1	1	1
Prod.string index:	2	2	2	2	2	2	2	2
Serial number index:	3	0	0	0	0	3	3	0
Number of conf.:	4	1	1	1	1	1	1	1
Configuration descriptor:								
Total length:	27	22	003B	22	54	64	35	00C1
Num.intf:	1	1	2	1	3	3	1	1
Conf.value:	1	1	1	1	1	1	1	1
Conf.string:	5	0	4	0	0	0	0	0
Attr.:	C0	80	A0	A0	A0	A0	80	80
Max.pwr:	FA	FA	31	32	32	FA	E1	50
Interface descriptor:								
Intf.number:	0	0	0	0	0	0	0	0
Alt.:	0	0	0	0	0	0	0	0
Endpoints:	3	1	1	1	1	1	5	2
Intf. Class:	6	3	3	3	3	2	FF	FF
Intf. Subclass:	1	0	1	1	1	2	FF	FF
Intf. Protocol:	1	0	1	2	1	0	FF	FF
Intf.string:	0	0	0	0	0	0	5	0

Fig. 3. Part of the descriptors of some devices under test

The HID class has several subclasses, each processed in its own way. The VID and PID product IDs must be unique for the released product. An important descriptor is also the serial number of the device, it can be recognized and processed the same when reconnecting, even if a different physical USB port is used. It is important for storage devices to be assigned the same drive letter; for devices such as serial port adapters and modems, to be assigned the same COM port designation.

5. 2. Data structuring

Not all devices are recognized in the same way both in structure and in data content when read from different microcontrollers, which is confirmed by the data in Fig. 4.

12 devices have a diverse data structure, for example, the iPhone issues 301 descriptors, joystick – 43, keyboard – 62, mouse – 43, game mouse – 81, LilyPad Arduino – 99, USBmodem – 63, web–camera – 192, USB 3G modem – 45, Android-device 0, doc–station – 39, Mega Arduino – 45, different devices – different number of descriptors and different structural completeness, which is reflected in their quantitative analysis in Fig. 5.

iphone		joystick		keyboard		mouse	
Interface descriptor:		Interface descriptor:		Interface descriptor:		Interface descriptor:	
Intf.number:	0	Intf.number:	0	Intf.number:	0	Intf.number:	0
Alt.:	0	Alt.:	0	Alt.:	0	Alt.:	0
Endpoints:	3	Endpoints:	1	Endpoints:	1	Endpoints:	1
Intf. Class:	6	Intf. Class:	3	Intf. Class:	3	Intf. Class:	3
Intf. Subclass:	1	Intf. Subclass:	0	Intf. Subclass:	1	Intf. Subclass:	1
Intf. Protocol:	1	Intf. Protocol:	0	Intf. Protocol:	1	Intf. Protocol:	2
Intf.string:	0	Intf.string:	0	Intf.string:	0	Intf.string:	0
		Unknown descriptor:		Unknown descriptor:		Unknown descriptor:	
Endpoint descriptor:		Length:	9	Length:	9	Length:	9
Endpoint address:	2	Type:	21	Type:	21	Type:	21
Attr.:	2	Contents:	1E+17	Contents:	1E+17	Contents:	1,1E+17
Max.pkt size:	40						
Polling interval:	0	Endpoint descriptor:		Endpoint descriptor:		Endpoint descriptor:	
		Endpoint address:	81	Endpoint address:	81	Endpoint address:	81
Endpoint descriptor:		Attr.:	3	Attr.:	3	Attr.:	3
Endpoint address:	81	Max.pkt size:	8	Max.pkt size:	8	Max.pkt size:	4
Attr.:	2	Polling interval:	0A	Polling interval:	0A	Polling interval:	0A
Max.pkt size:	40						
Polling interval:	0						

Fig. 4. Different data structure of USB devices under test

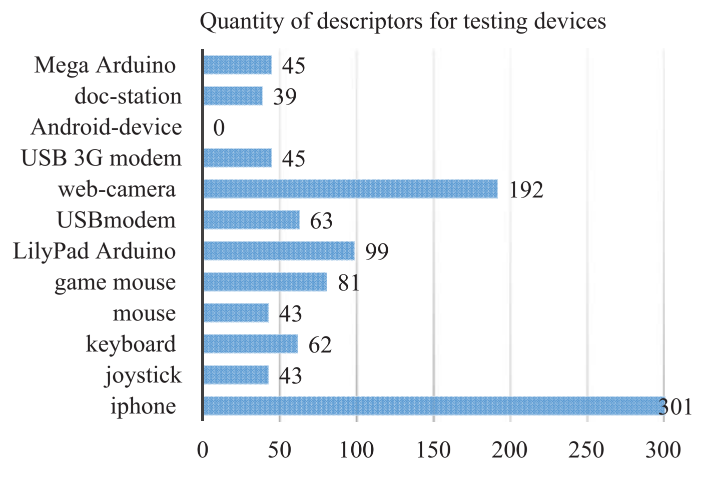


Fig. 5. Quantitative analysis of USB device descriptors read

Fig. 5 displays the names of USB devices along the vertical axis, and the number of received descriptors from the corresponding device along the horizontal axis. The USB standard defines a number of descriptors that are common to all classes of devices, and they are read by a request Get_Descriptor that uses the descriptor code. And although device, configuration, interface, endpoint descriptors are required, devices do not always issue even them.

5. 3. Consistency of device modes of operation

All descriptors share a common format. The first byte indicates the length of the descriptor in bytes, the second byte indicates the type of descriptor. The device descriptor specifies some information about the device. For example, the supported USB version, the maximum package size, vendor, and product IDs (VID and PID), and the number of possible configurations that the device can have. Most devices are simple and have only one configuration. The configuration descriptor indicates how the device is powered, what its maximum power consumption is, and the number of interfaces that the device has. Therefore, it is possible to have 2 configurations for the device – one for bus power, the other for power from an external source. Since this is the header to the interface descriptors, it is also possible to have different configurations for different transfer modes.

There are many descriptor settings. In order for the work on the recognition of the device to proceed adequately, one needs to most fully learn information about the device. To do this, a Python script was written and loaded onto the Arduino Mega board, since through it the interaction of the operating system kernel and the USB device was carried out. Knowing the entire list of combinations of descriptor parameters, data for all tested devices were obtained.

According to the specification of USB devices, three modes of operation are regulated: Low-Speed (10-1500 Kbps); Full-Speed (0.5–12 Mbps) High-Speed (25–480 Mbps). Distribution of test devices by operating modes (iPhone – High-Speed, joystick – Low-Speed, keyboard – Low-Speed, mouse – Low-Speed, game mouse – Low-Speed, LilyPad Arduino – High-Speed, USBmodem – High-Speed, web-camera – Full-Speed, USB 3G modem – High-Speed, Android-device – High-Speed, doc-station – High-Speed, Mega Arduino – High-Speed) is shown in Fig. 6.

The next step is to analyze the type and amount of data that the device transmits to the operating system. For example, if coordinates are transmitted from a device, then most likely the USB device is a mouse. In Fig. 6, the horizontal axis displays the names of USB devices, and the vertical axis – the speed of operation and mode of the corresponding device.

As a result of our work, different devices were analyzed by class, according to the parameters and speeds of work, it is possible to identify the basis for creating a conclusion on the group of parameters about the possible attribution characteristically of the device with any class. However, the system does not always work correctly with the hardware and all device descriptors can be represented in a single form.

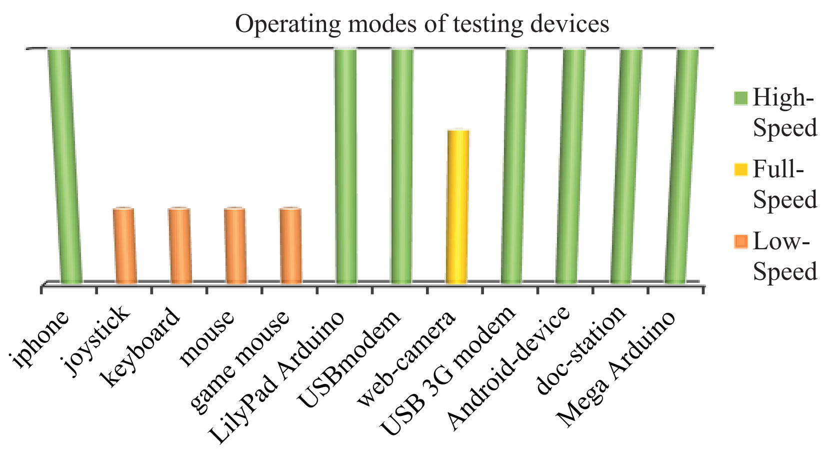


Fig. 6. Operating modes of the USB devices under test

6. Discussion of the work of the software and hardware system

The research was conducted through a software package implemented in the Arduino IDE environment. Physically, it is implemented on the Arduino Mega board with Shield, which reads the parameters of the devices (Fig. 1). With it one can track the launch of pre-modified devices and transfer HID descriptors from the connected hardware. The results could be used in deciding whether there is a pre-modification and serve as a mechanism (tool) to limit the functioning of possible malicious software from a connected USB device and install protection of information systems from attacks such as BADUSB. This measure increases the level of security of work with peripherals, cutting off attempts by external devices to take control of the entire system.

The described approach of comparing parameters and isolating multiple interface descriptors in single-function USB devices (Fig. 2) has been used more than once in experiments; that has made it possible to qualitatively analyze quantitative data sets.

To work with the data coming from the software and hardware system, the methods and algorithms for the functioning of USB devices were investigated. The obtained results (Fig. 3, 4) showed the presence of different structuring of data and the functioning of the devices themselves under different modes of operation (Fig. 6).

The experiments were conducted on Linux and Windows operating systems with 40 devices. The data allow the device driver for the USB bus itself to efficiently request a new connected device and wait for a response. The study was limited to 12 different classes of devices with different characteristics and, as a promising direction of development, it is possible to expand the fleet of tested devices. When one connects devices to the USB bus, each device signals its existence and reports the manufacturer ID and device ID without fail. These identifiers are the determining information when selecting a loadable driver, information about which is searched in the registry. The descriptors of 12 devices were described, their quantitative state is shown in Fig. 5, which means that a different number of USB device descriptors were readable. Figures showed parts of the data; we discussed the modes of operation of USB devices.

The resulting solution functions under the circumstances that the choice of components for the creation of the module was based on the technical characteristics of the boards, in order to check the operability of the hardware and software module, USB devices with modified firmware were implemented.

The developed software and hardware system receives data from USB-drives, determining the number of parameters and the parameters themselves in each case. A distinctive feature of the study is the implementation of the system on an external device with additional memory expansion on the Arduino Mega board with Shield. And unlike the Pro Micro board in [4], the system is implemented through libraries with increased data processing time. In the future, the strategy of introducing white lists is considered, as stipulated in [7]. One of the features is the large set of data obtained from 12 commonly used devices. Thus, the results are clearly presented in tabular form in Fig. 3, 4.

The results of our study could be used as a tool to limit the activities of USB data sources and install protection of information systems from BADUSB attacks. This measure increases the level of security of operating systems as a whole, cutting off attempts from external devices to work and execute unauthorized requests for programs, starting and stopping other programs, entering and output data, allocating and releasing additional memory, loading programs into RAM and their execution. And this is what makes it possible to maintain a decent level of security and recognize a possibly pre-modified device at the stage of connecting to the system.

These studies could be used as part of the protection against possible threats of BADUSB both in the lives of users within the home, and as a separate protection module in the office, as well as where there is a need to connect many USB devices and guarantee the safety of their functioning. At the same time, it is necessary to connect the devices in turn and wait for the final decision on each of the devices from the system.

The direction of further development may be the expansion of the polygon of tested devices, as well as the study of the possibility of creating independently functioning software in Linux and Windows operating systems.

7. Conclusions

1. At the first stage, when extracting descriptors from the devices of the test range, the information was represented in a selective form according to the data on 8 devices: iPhone, joystick, keyboard, mouse, game mouse, LilyPad Arduino, USBWiFimodem, web-camera. Information on device descriptors is presented on different interfaces, each interface consists of one or more alternative parameters, and each alternative parameter consists of a set of endpoints. A configuration descriptor describes the entire configuration (device capabilities), including its interfaces, alternate settings, and endpoints. Each of these entities is also described in their descriptor format. A configuration descriptor sometimes includes custom descriptors defined by the device manufacturer. However, regardless of the devices, the total length of the device descriptor is 12 bytes. Some devices, such as the iPhone, LilyPad Arduino, USB WiFi modem in the USB version parameter showed that they work according to the USB 2.0 standard. The rest work with the 1.1 standard. This is due to the different release times of the devices, which makes it possible to use different data transfer technologies.

2. The completeness (structuring) of the data of certain devices showed heterogeneity both in the length of the main descriptors and in their content. The information provided for configuring devices was generated at the request of the device polling module: device information, information on device descriptors, configuration descriptors and interfaces.

According to the data obtained, all devices are represented by a different number of descriptors and the best described in this regard is the Apple device (301), and the worst – Android device (0). This is due to the imprudence of the manufacturer in providing the characteristics of the equipment provided. This allows for and gives a large field of opportunities for an attacker to substitute any identifiers as native in practical application.

3. In our studies, different devices were analyzed by class and, according to their modes (parameters and speeds of operation), it is possible to synthesize patterns of work on a group of output parameters about the possible correlation of devices with any class and restrictive capabilities. The peculiarity is that it is impossible to classify all devices in the same way for various reasons (different manufacturers and different approaches to systematization, descriptive content of descriptors, etc.). The number of possible configurations that the device can have is determined; most devices are single-functional and have 1 configuration but it is also possible to have different configurations for different data transfer modes.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was performed without financial support.

Data availability

The data will be provided upon reasonable request.

References

1. Neuner, S., Voyiatzis, A. G., Fotopoulos, S., Mulliner, C., Weippl, E. R. (2018). USBlock: Blocking USB-Based Keypress Injection Attacks. Lecture Notes in Computer Science, 278–295. doi: https://doi.org/10.1007/978-3-319-95729-6_18
2. Yang, B., Qin, Y., Zhang, Y., Wang, W., Feng, D. (2016). TMSUI: A Trust Management Scheme of USB Storage Devices for Industrial Control Systems. Lecture Notes in Computer Science, 152–168. doi: https://doi.org/10.1007/978-3-319-29814-6_13
3. Johnson, P. C., Bratus, S., Smith, S. W. (2017). Protecting Against Malicious Bits On the Wire. Proceedings of the 33rd Annual Computer Security Applications Conference. doi: https://doi.org/10.1145/3134600.3134630
4. Ramadhanty, A. D., Budiono, A., Almaarif, A. (2020). Implementation and Analysis of Keyboard Injection Attack using USB Devices in Windows Operating System. 2020 3rd International Conference on Computer and Informatics Engineering (IC2IE). doi: https://doi.org/10.1109/ic2ie50715.2020.9274631
5. Mueller, T., Zimmer, E., de Nittis, L. (2019). Using Context and Provenance to defend against USB-borne attacks. Proceedings of the 14th International Conference on Availability, Reliability and Security. doi: https://doi.org/10.1145/3339252.3339268
6. Karystinos, E., Andreatos, A., Douligeris, C. (2019). Spyduino: Arduino as a HID Exploiting the BadUSB Vulnerability. 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). doi: https://doi.org/10.1109/dcoss.2019.00066
7. Mohammadmoradi, H., Gnawali, O. (2018). Making Whitelisting-Based Defense Work Against BadUSB. ICSDE'18: Proceedings of the 2nd International Conference on Smart Digital Environment. Available at: http://www2.cs.uh.edu/~gnawali/papers/badusb-icsde2018.pdf
8. Ji, X., Le Guernic, G., Cuppens-Boulahia, N., Cuppens, F. (2018). USB Packets Filtering Policies and an Associated Low-Cost Simulation Framework. Lecture Notes in Computer Science, 732–742. doi: https://doi.org/10.1007/978-3-030-01950-1_44
9. Hernandez, G., Fowze, F., Tian, D. (Jing), Yavuz, T., Butler, K. R. B. (2017). FirmUSB. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. doi: https://doi.org/10.1145/3133956.3134050
10. Pyrkova, A., Zuyeva, Ye. (2019). Creating BADUSB devices and system safety analysis. Vestnik KazNITU, 5, 466–470. Available at: https://official.satbayev.university/download/document/12327/%D0%92%D0%95%D0%A1%D0%A2%D0%9D%D0%98%D0%9A-2019%20%E2%84%965.pdf
11. Zueva, E. A., Pyrkova, A. Yu. (2019). Nvestigation of USB devices using ducky script. Vestnik AUES, 3, 53–57. Available at: https://vestnik-aues.kz/frontend/web/uploads/magazine/pdf/1591966671_nFx8A8.pdf#page=55
12. Zueva Ye. (2020). Analysis of work of devices with BADUSB vulnerability. Vestnik KBTU, 17 (1), 141–146. Available at: https://kbtu.edu.kz/images/vesnik_1_2020.pdf