Making Whitelisting-Based Defense Work Against BadUSB

Hessam Mohammadmoradi, Omprakash Gnawali
University of Houston
{hmoradi,gnawali@cs.uh.edu}

CCS Concepts

•**Security and privacy → Intrusion/anomaly detection and malware mitigation;**

Keywords

USB Malware; BadUSB; Whitelist

ABSTRACT

Universal serial bus (USB) devices have widespread use in different computing platforms, including IoT gadgets, but this popularity makes them attractive targets for exploits and being used as an attack vector by malicious software. During recent years, several reports [17] ranked USB-based malware among top 10 popular malware. This security flaw can slow down the increasing penetration rate of IoT devices since most of those devices have USB ports. The research community and industry has tried to address USB security problem by implementing authentication protocols to protect users' private information and also scanning USB's storage space for any malicious software using their own repository of malware signatures, or simply disallowing use of USB devices on desktops. The new generation of USB malware does not hide in storage space, which means they are not detectable by conventional anti-malware. BadUSB is a malware recently introduced by security researchers. BadUSB modifies USB firmware and can attack all the systems which the infected USB is plugged in. The only applicable solution against this new generation of malware is whitelisting. However, generating a unique fingerprint for USB devices is challenging. In this paper, we propose an accurate USB feature based fingerprinting approach which helps us to create a list of trusted USBs as device whitelist. Our solution prevents and detects BadUSB and similar attacks by generating fingerprint from trusted USB devices' features and their primary usage. We verified the uniqueness of our generated fingerprints by analyzing real data which is collected from USB drives used by students in academic computer labs over one year. Our results indicate that our feature based whitelisting approach with an accuracy of 98.5% can identify USB whitelist members.

ICSDE'18, October 18–20, 2018, Rabat, Morocco

DOI: https://doi.org/10.1145/3289100.3289121

1. INTRODUCTION

Universal Serial Bus (USB) is one of the most popular communication standards supported by many computing and IoT devices. Many IoT gadgets allow data copy, firmware and configuration update using USB port. The main advantages of USB devices are convenience, reliability, and being powered directly from the USB port. At the same time, this popularity also makes them one of the most widely available vectors for malware propagation or host infection. In fact, USB devices have been successfully used as vector for many USB attacks. In Microsoft's security intelligence report published on Dec 2015[17], VBS/Jenxcus, a worm that gives control of the computer to the attacker and spreads by infected removable drives, like USB flash drives, has been positioned in the top 10 reported Malware from Windows machines all around the world.

Recently Karsten Nohl and Jakob Lell from Security Research Labs announced a new type of USB malware called BadUSB [22].Their patched firmware can emulate a keyboard, spoof a network card, and even spoof a display. Usually, anti-malware solutions scan contents stored on USB devices' storage space but the firmware is always assumed to be trusted. One approach to detect infected firmware is by requiring all USB to have firmware signed by the manufacturer and having the host check the firmware certificate. This solution, however, is hard to implement on the large number of devices which are already in the market and requires major reengineering in a large number of systems, spanning devices and OS. Almost all the IoT devices we use inside homes or offices have a USB port which makes them potential targets for attackers. To the best of our knowledge, the practical solution against BadUSB is creating a whitelist of trusted USB devices. In this approach, all USB devices are assumed suspicious, except those that have been registered in the whitelist. Then, a firewall can block unlisted devices or enforce rigid security constraints on them.

To create a whitelist of trusted devices, each device should be identified with a unique fingerprint. Current USB whitelisting approaches [6, 16] use features like serial number and product ID to uniquely label each USB device but our study shows that serial numbers are not unique. Further, there are many USB devices such as Webcams and keyboards that do not provide any serial number. Multiple instances of the same USB device (e.g. same brand Webcams) share same

product ID and vendor ID which means they are not distinguishable by conventional whitelisting solutions.

In this paper, we suggest an approach that successfully detects and prevents BadUSB attacks using feature-based whitelisting approach. We generate a unique identifier for each USB device and build a whitelist of USB devices. Our key insight comes from how BadUSB attack works – the malware in the known but infected device changes the primary service of the device to a different service (For example, present a Storage device to the host system as a Network device). Thus, the device fingerprint, in addition to ID-related features, should also include the primary service of the device. However, this approach requires stronger (than what can be provided by USB product ID) ID-related feature first, which can be augmented with primary service field to detected BadUSB. Combining product ID and vendor ID with other features like firmware version, USB interface, USB Class and driver version generates a more unique fingerprint for USB devices.

We evaluated our suggested fingerprinting technique over existing USB devices and our system can uniquely identify a device with 98.5% accuracy. Our contributions in this research are:

- Propose an effective feature-based whitelisting against reflashing attacks like BadUSB and BadAndroid. We generate unique fingerprints for USB devices that includes their primary services and use the fingerprint to allow authorized devices that are not infected by BadUSB.

- Reveal the unreliability of current USB whitelisting solutions: Current solutions for USB whitelisting rely on properties such as serial numbers which our investigation shows are not as unique as they are assumed to be and are unavailable in USB devices such as keyboards and Webcams. Besides, BadUSB will not change properties like serial number or vendorID which means current whitelisting approaches will not notice BadUSB attack on one of the trusted devices.

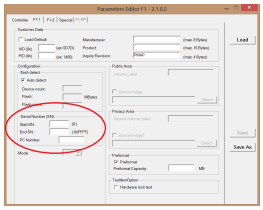
- Show how to generate unique IDs for USB devices to enable whitelisting-based defense: We propose fingerprinting technique for USB so that USB devices can be assigned unique IDs. Our fingerprinting technique can generate IDs even for USB devices that do not have serial IDs, e.g., keyboards and Webcams.

- Verify accuracy of the proposed method using real data: We collected data from USB devices used by students over one year. We used that data to verify the accuracy and applicability of our approach to build effective USB whitelist using features collected from them.

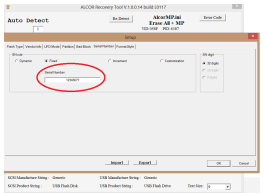
- Validate effectiveness of our approach via BadAndroid: BadAndroid is proof of concept implementation proposed by BadUSB inventors. We deployed BadAndroid on a cell phone and describe how we can detect BadAndroid infection.

2. NEW GENERATION OF USB THREATS

New generation of USB based threats are called reflashing attacks in which malicious software is flashed directly



(a) KingStone



(b) Alcor

Figure 1: Recovery Tools Allow User to Overwrite Serial Number

onto the chip of the USB device. In other words, USB drive keeps working as expected and data can be transmitted to and from the device but there are some unauthorized actions silently running in the background. To reflash the USB, attacker can completely delete original firmware and copy his own developed software or just append his code to existing firmware. The second approach is very hard to detect. As an example, a USB mass storage device can act like an HID (Human Interface Driver) and after connecting to system emulate keystrokes and download a keylogger to target system.

BadUSB which was first disclosed by security researchers in July 2014[22] is an example of the new generation of USB threats. BadUSB exploits a major security hole in the design of USB devices which allowed its inventors to spoof network card, redirect Internet traffic and install additional malware. The BadUSB hides on the firmware section of the device instead of storage space. In more detail, conventional anti-malware cannot detect BadUSB and even formatting the device will not remove BadUSB.

BadUSB attack is a specific version of the USB reflashing attacks [22]. In addition to its actual attack payload the attack varies within the following scenarios [22]:

- Windows takeover: USB drive emulates keystrokes to send command to Windows. As long as USB drive remains connected to the system it can send unauthorized commands to operating system.

- Sniff Root Password: Attacker reflashes USB drive with malfunction software. Once the infected USB drive is plugged to a machine with Linux operating system, root password which inserted by user during screen unlock is sniffed by USB.

- Cellphone Attack: Once infected Android phone is connected via USB drive, it starts modifying DNS table and redirects specific URLs which ask for user's financial or personal information (Paypal as an example) to IP's controlled by attacker.

- USB Boot Sector Virus: BadUSB can install malfunctioned BIOS by emulating keystrokes and gain control of entire operating system on each boot.

2.1 Current Defense Mechanisms

First possible solution is locking USB devices so they will not accept firmware which is not signed by valid manufacturer [7]. If only the manufacturer can issue firmware upgrades, it will exclude software and contributions from the open source community to keep the USB ecosystem open. Ultimately, signed firmware and authentication of firmware

for all USB devices will require major reengineering of a large of number existing systems and the appetite for such efforts among the USB device, OS developers, and computer manufacturers is unclear. Another approach, applicable for the devices already on the market, to protect USB firmware is creating a whitelist of USB devices. GData [5] firewall notifies user once a new device is plugged into system and asks user to label it as whitelist or blacklist device. Each USB device has a pair of information about its vendor (VendorID,ProductID) and GData uses this information to create a unique ID for each USB device. Problem with using VendorID/ProductID pair for uniquely identifying devices is that similar devices manufactured by the same vendor will have same vendorID and same product ID.

Another attribute can be used for fingerprinting is serial number. There are multiple problems regarding serial numbers. First of all, 54% of our records from USB devices used by students do not have serial numbers. Devices like mouse's, Webcams and keyboards do not provide valid serial numbers. Even if we limit our problem to devices with valid serial numbers (USB Mass Storage Devices), still problem would not be solved. Major USB manufacturers developed lightweight utilities which update USB firmware. These tools help costumers to repair their damaged devices. Figure 1 shows screenshots of two popular ones (Alcor and Kingstone). We conducted a small survey about current whitelisting solutions available commercial or as open source products and tried to find their method of fingerprinting USB devices. The results have been summarized in table 1.

Based on facts regarding the uniqueness of serial number, product id and USB class and also table 1, the conclusion is there is no effective USB whitelisting approach available right now.

3. DEFENCE

3.1 Attack Model

Our main focus in this paper is about reflashing attacks on USB devices. In these attacks, the firmware of USB device will be patched with malicious code. Patched firmware changes device's default service and spoofs other services than original service. For example, a USB mass storage device pretends to be a keyboard and sends keystrokes to operating system. *We assume a strong adversary*: the attacker has physical access to a white-listed USB device and is able to copy all the device parameters (e.g., those listed on Table 2) to the USB device to be used to launch the attack.

3.2 Trust Model

In this section, we propose our approach to detect BadUSB and similar USB flashing attacks. One of the possible solutions against BadUSB is whitelisting, but, uniquely identifying each USB device seems to be a challenging in current USB whitelisting approaches. We propose a dependable approach to uniquely identify each USB device, we can create a list of trusted USB drives and block all other USB devices or generate an alarm once the connected USB device does not belong to the whitelist. Large scale organizations can build their own whitelist of trusted USBs inside the company. Even if the attacker somehow finds access to trusted devices, and emulates or spoofs trusted device's features, in attacks such as BadUSB which change the device's original

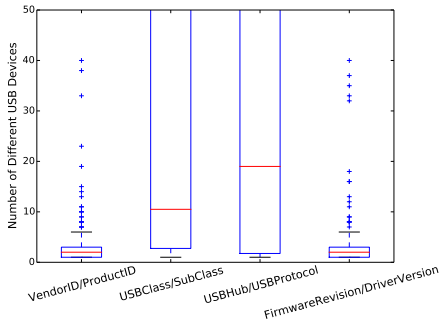


Figure 2: Number of Different Devices with Same Value for a Specific Feature Subset

interface, this unauthorized change will be detected by our firewall.

To build the whitelist of trusted devices, a unique identifier should be assigned to each USB device; but, such a unique identifier available across whole USB devices does not exist. In the following sections, we briefly analyze potential nominees to be unique identifiers for USB devices.

3.3 Potential Unique Identifiers

We have collected many features but which ones can be used to generate unique identifiers? In this section, different possible candidates have been analyzed.

3.3.1 Serial Number

As we explained in previous sections, serial numbers are not available in almost half of USB devices and also easy to spoof.

3.3.2 DeviceInstanceID

Windows generates a unique ID for each connected device which is called Device Instance ID [3]. A device instance ID is a system-supplied ID that uniquely identifies a device in the system. Device instance ID consists of Instance ID and device ID.

A device instance ID is persistent across system restarts. According to Microsoft's official website[3]: "An instance ID is a device identification string that distinguishes a device from other devices of the same type on a computer. An instance ID contains serial number information if supported by the underlying bus, or some kind of location information. A device ID is a string reported by a device's enumerator".

DeviceInstanceID is not a wise choice for whitelisting. It is unique per system but it can not uniquely identify one device across the whole network. In our collected data almost 10% of devices with unique serial numbers have more than one assigned InstanceIDs. In other words, the same device reported by at least two different desktops with two different InstanceIDs which means whitelist of InstanceIDs will not work over the network of organization's machines.

3.3.3 Feature Sets

Based on previous sections, it seems separate features can not uniquely identify USB drives. How about a set of features? Set of features together is useful if they can uniquely identify USB devices.

We selected sets of USB features and counted the num-

Table 1: State of the Art USB Whitelisting Solutions

Name	License	Features used to generate fingerprint
Solar Wind [6]	Commercial	Instance ID
GFI [16]	Commercial	Serial Number , USB Class
Windows Server Firewall [2]	Commercial	Make, Model & Revision of Device
Symantec Data Loss Prevention [10]	Commercial	Product ID, Vendor ID & Serial Number
USB Guardian [15]	Open Source	Serial Number, USB Class, Vendor ID & Product ID
USB Guard [14]	Open Source	Serial Number
Lumension End Point Security [9]	Commercial	InstanceID & USB Class
EverStrik USB Whitelisting [4]	Commercial	Product ID, Vector ID & Serial Number
Guardian - USB Whitelisting Script[11]	Open Source	Vector ID, Product ID, Device Description

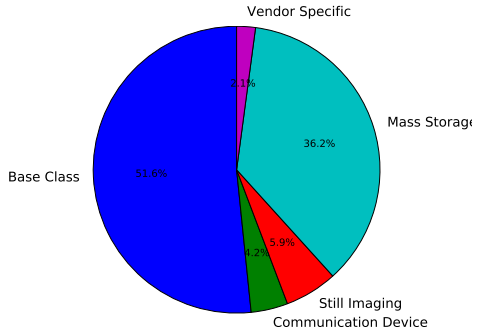


Figure 3: USB Class Codes Share in Collected Dataset

ber of devices in our dataset which have similar values for each set; These feature sets have been selected from feature sets used in current USB whitelisting approaches (Table 1). Results are shown in figure 2. For example, in figure 2 VendorID/ProductID subset has an average value of 3 which means in our dataset for each pair of (VendorID,ProductID) in average, there are three different devices. The final conclusion from figure 2 is that none of the feature sets which already are being used by other whitelisting approaches can create a reliable whitelist of USB drives.

Another possible identifier combination is USBClass, USBSubClass, USBProtocol. Each USB device defines class code information which is used by operating system to identify a device's functionality and to nominally load a device driver based on that functionality. The information is contained in three bytes with the names Base Class, SubClass, and Protocol.

Figure 3 illustrates share of each USB class in total plugged in USBs in our dataset. The Base class generally used by general devices are connected to USB port such as keyboard and monitor. This class is about 51% of total records. Second most claimed class is Mass Storage class. Apple and Samsung companies usually use Imaging and Communication classes for their USB devices. Figure 3 illustrates that both of them have the same share in our dataset. There are also some vendor specific USB devices in our records. Figure 3 proves thatUSBClass, USBSubClass, USBProtocol set can not uniquely identify USB devices.

3.4 Proposed Unique Identifier

Our solution to generate a unique fingerprint for each USB drive is building a model based on all the features collected from USB devices.

Our early analysis revealed that there are high values of correlations between multiple pairs of features. To improve efficiency of the fingerprinting process, from each pair of highly correlated features, just one feature is selected. After removing redundant features, final feature vector contains 24 features including DeviceType, VendorID, ProductID, USBClass, DriverFileName and USB protocol.

4. BUILDING OUR DATASET

This section elaborates our data collection procedure and gives summary information about collected dataset. Our goal in this phase was collecting usage information and device properties from USB drives used by people; Having that dataset, we can evaluate the performance of our USB malware prevention and detection approach.

4.1 Data Collection Procedure

Our monitoring agent is a lightweight Java application that reads information from Windows registry file and sends it through the network to our central database server. Windows saves a registry record each time a USB device is plugged into or plugged out of each USB port [13]. We installed our application on two different academic labs located on two different floors. One of the labs is primarily used for course related sessions and the other one is used by students for general purposes. One of the challenges we had to solve was the inaccuracy of Windows in timestamping plug in and plug out events. All events recorded in Windows registry have a timestamp but this timestamp for USB devices is not persistent during restarts. To solve this problem, our data collection script timestamps all the events before sending them to database. Sampling frequency for our application is one sample per minute which keeps granularity of samples in reasonable level and also will not degrade performance for host systems or lab's local network (We installed our application on all the systems and each application sends data to the server every one minute).

4.2 Collected Features

During data collection phase, we collected a set of features to uniquely identify and distinguish each USB device. First, we started by collecting serial numbers. Each USB mass storage device provides a serial number during its initial registration on operating system.

Different USB drives have their own communication protocols and use various operating system services. Our application collects USB protocol, driver version, and manufacturer ID. We also were interested in measuring length of time

Table 2: Collected Features

Serial Number	Device Name	Device Description	Driver Letter	Vendor ID
Firmware Revision	Hub	Vendor Name	Product Name	Product ID
Parent ID	Service Name	Service Description	Device Mfg	Driver Filename
Power	USB Version	Driver Description	Driver Version	USB HUB
InstanceID	Safe To Remove	USBProtocol	IP	Protocol Description
Creation Time	Last Plug/Unplug Time	Username	MAC	Driver Version

each USB device remains connected, so, each time a USB device is plugged into a computer, our application sends a plug in message to the central database server; when that device is unplugged, again our monitoring application sends a message to the server indicating disconnection event. We have every plug and unplug event in our database and we can calculate total time each device remains connected. Totally we collected 30 different features about each USB device. Here are short descriptions for some of more important features.

- Device Type: The device type, according to USB class code [12]. For instance, we have USB mass storage and USB input and output devices as most popular types of USB devices.
- Serial Number: Specifies the serial number of the device. This attribute is only relevant to mass storage devices (Flash memory devices, CD/DVD drives, and USB hard-disks).
- Service Name & Description : All the USB devices need to register their primary service during enumeration phase. Examples are USB Mass Storage or HID.
- Created Date: Specifies the date/time that the device was installed. This date/time value represents the first time, user plugged the device into the USB port.
- Last Plug/Unplug Date: Specifies the last time that user plugged/unplugged the device.
- VendorID/ProductID: Specifies the VendorID and ProductID of the device. Using the table of producers IDs [8], easily we can recognize the manufacturer of each device.
- USB Class/Subclass/Protocol: Specifies the Class/ Subclass/Protocol of the device according to USB specifications.
- Username:When we have username for each USB record, we can map each USB device to his owner and use this map to analyze USB users behavior.
- Device InstanceID: Windows uses device's serial number and some other location information to generate Device InstanceID which is a unique identifier among devices are connected to the system.

Table 2 contains list of other collected features.

4.3 Monitoring Environment Characteristics

We collected data from academic computer labs. Table 3 shows monitored environment's characteristics. All of our results and conclusions are extracted from data collected during November 2013 till December 2014. All of the systems were using Window 7 or Windows 8 as their operating system. Our application ran on each individual system and watched USB ports for plug and unplug events. Besides information about USB device, our application also sends IP address, MAC address of the machine and hashed version of logged in user's username (to preserve user's privacy we did not collect usernames). Using machine information we can track USB devices during data collection period.

Table 3: Summary of USB Usage Data Collection Environment.

Number of monitored systems	57
Monitoring Interval	Nov 2013 - Dec 2014
Recording Interval	1 minute
Host Operating System	Windows 7 & 8
Number of USB Events	8607
Number of Unique USB Devices	596

After almost 1 year of data collection, we have 8607 records from diverse range of USB devices such as Mass Storage Devices, Keyboards, Webcams and cell phones. After careful investigation in our database, we counted 596 unique USB devices.

5. EVALUATION

In this section, we evaluate accuracy of our proposed fingerprinting approach in uniquely identifying USB devices and distinguishing whitelisted devices from unauthorized ones. We also run BadAndroid against our whitelisting approach to evaluate our solution's effectiveness in protecting systems against famous BadUSB implementations. In the rest of this section, identification accuracy means percentage of USB devices among all collected USB devices which receive unique identifiers based on combining selected feature set.

5.1 Feature Ranking

After manually removing highly correlated features, we reduced our original feature set to 24 features which are collected from USB devices. Do we really need all those 24 features to be able to uniquely identify each USB? In this section, we tried to rank features based on their information gain value. We used Weka information gain calculator to measure each feature's information gain.

As we expected and also illustrated in figure 4, different features have different importance based on the amount of information they reveal. General numerical values such as ProductID, VendorID and FirmwareRevision have larger information gain compared to categorical values such as DeviceType. Another interesting insight from figure 4 is UBS class and subclass values have been ranked in lower positions which indicates they can not be a reliable source for USB identification. Figure 5 illustrates the importance of feature information gain on final identification accuracy. To plot

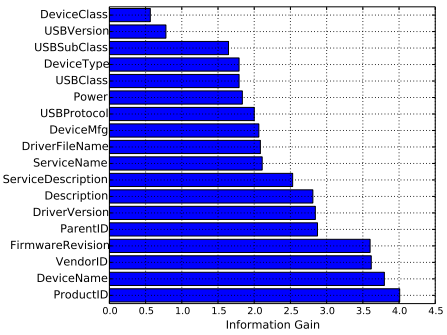


Figure 4: Information Gain Value Per Feature

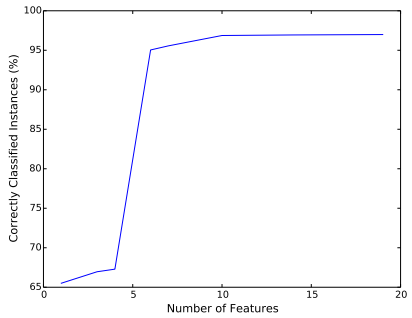


Figure 5: Feature Ranking Impact on Fingerprinting Uniqueness

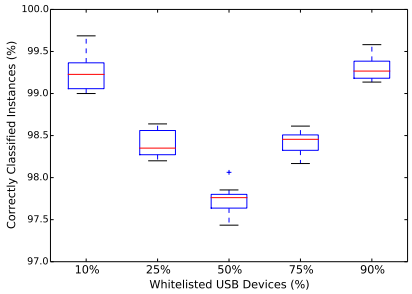


Figure 6: Accuracy of Detecting Unauthorized Devices

figure 5, we sorted features based on their information gain and added the top-ranked feature to selected feature set. Then, we calculated identification accuracy values based on the selected feature set, walked through the entire sorted feature list and added features one by one to previously selected feature set. Our selected feature set finally covers all the collected USB features. The whole process is illustrated in algorithm

5.2 Overall Performance Evaluation

To evaluate the accuracy of the proposed solution, we randomly selected a subset of USB devices from our dataset and labeled them as whitelisted USBs and rest of devices were tagged unauthorized. We performed five rounds of experiments for each subset size. We used all 24 features in our whitelisting approach to distinguish whitelisted and unauthorized devices. Figure 6 shows number of correctly identified in each round.

Based on results in figure 6, our solution will distinguish whitelisted devices from unauthorized ones with an accuracy of 97.5% in the worst case. Further, the accuracy of our approach varies in the range of 97.5 % to 98.5% across the different number of whitelisted devices in our dataset.

5.3 Detecting Bad-Android

Now we describe how we implemented and launched a real BadUSB attack on our test system and describe how we can defend against the attack. The inventors of BadUSB implemented BadAndroid as a proof of concept for their attack [1]. BadAndroid script runs inside Android phone and spoofs a USB-Ethernet adapter from it. Thus, when the phone is connected to the target machine via USB port, the spoofed USB network interface becomes available to machine and the target machine starts using the spoofed network interface. Thus, the infected device is able to capture network traffic from target computer. In next step, BadAndroid changes *some* DNS replies to redirect IP traffic – the attacker creates a hosts file on the phone which includes a list of URLs from which the attacker desires to capture the traffic.

To evaluate performance of our proposed approach, we first rooted a Samsung Galaxy S2 (SGH I727) cell phone running Android 4.1.2. Then we installed BusyBox emulator followed by an installation of BadAndroid script. Next, we use a simple scenario to launch the attack – just connect one infected device to a computer. After connecting the cellphone to the USB port, it registered itself as Ethernet card and provided false DNS answers to the machine for our pre-selected URLs (www.paypal.com in our case). The attack was successful on Windows 10 and Ubuntu 16.4 systems with all security patches installed.

Table 4 shows features we collected before infection and after infection.

As seen in table 4, ”device type” feature changed from ”UnKnown” to ”Remote NDIS”. After BadUSB infection, at least one of the Device's features changed which means its fingerprint also will change and our approach will detect the change and block the USB. While state of the art whitelisting approaches (Table 1) can not detect this attack because the features they use to identify devices (Serial Number, Vendor ID and Product ID), did not change before and after the infection. Earlier solutions like GoodUSB also would not suspect these changes on a device such as cellphones or other devices that have vendor-specific device description field.

6. RELATED WORK

Whitelisting always has been one of the safest options to protect computing systems against malicious software tools and activities. A lot of research has been conducted in Phishing detection area to find effective blacklisting and whitelisting approaches [19, 18, 23]. Phishing attacks are mostly about stealing user's financial information such as credit card information and bank account passwords. For each user there are few websites which are authorized to ask for personal credentials and a whitelisting approach perfectly works in this situation. The critical point in this type of whitelisting is feature set which is selected to generate fingerprints. Features should be able to uniquely identify a

Table 4: Cellphone Features Before and After BadAndroid Infection- Changed features are in bold font

Feature	Before Infection	After Infection
Device Name	SAMSUNG-SGH-I727	SAMSUNG-SGH-I727
Device Description	SAMSUNG Mobile USB Composite Device	SAMSUNG Mobile USB Composite Device
Device Type	**Unknown**	**Remote NDIS**
Serial Number	1073dde	1073dde
VendorID	04e8	04e8
ProductID	6863	6863
Firmware Revision	**4.00**	**2.31**
USB Class/SubClass/Protocol	**00/00/00**	**e0/01/03**
Instance ID	USB\ VID04E8 PID6863\1073dde	USB\ VID04E8PID6863\1073dde

website and also should not be spoofable by attackers.

Another area that has application for whitelisting is denial of service (DoS) attack protection[28, 18]. In DoS, the attacker's goal is disrupting network service by sending overwhelming requests to target server. One working solution, in this case, is creating a whitelist of most important clients in critical service provider's server and give the highest priority to whitelist members. Again collected features which used as whitelist members' identification should not be spoofable by attackers.

There are some researchers [25],who analyzed USB-based software attacks and protection solution. But all of those research is about old generation of USB malware which are mainly malicious software hidden in storage space or attacks to gain unauthorized access to private information. BadUSB is an effective attack which does not require deep knowledge which makes it much more dangerous and to the best of our knowledge, currently, there is no practical effective solution against it and we are the first group who propose accurate and reliable solution to detect BadUSB. The new generation of USB malware recently absorbed attention from research community and there are few approaches trying to propose prevention solutions against BadUSB[24]. [27, 26] suggested to restrict interfaces based on USB's primarily usage. The authors in [26] used user expectation as a filter to restrict USB drives access. Each time there is new USB device attached to the system, they ask the user to select primary service for that USB based on claimed USB class of device. They will save the device's ID and its user assigned access control. In the future connections, protection system will not allow the USB to use other interfaces than authorized ones. The first serious problem with this approach is USB identification. They did not propose any solution for uniquely identifying USB devices. Their solution is not applicable in large scale organization with hundreds of USB drives existing. Another problem is that they use USB classes to identify legitimate interfaces for USB devices. Based on figure 3, more than 50% of USB devices use base class code. For example keyboard or mouse. In this case, the malicious keyboard can claim legitimate interface and still spoof keystrokes. In addition, there are approximately 12% of USB devices in our dataset which they have vendor specific interfaces (mostly cellphones). As the authors of GoodUSB themselves also mentioned in [26], their solution is not working in the case that vendor specific interface is being used by the device. The fingerprint generated by our solutions also includes USB class and interface and if there is a change in these features, the new fingerprint will not match with original one. In addition to all previously mentioned problems with access control solutions, authors at [20] suggested a sniffing attack on the USB bus. They register their USB as a normal USB device and transparently eavesdrops all the downstream data from the host to all other devices connected to the BUS. In other words, this attack does not use any unauthorized interface and access control approaches like [26] can not protect the systems against these kinds of attacks. In our solution, if the device is not in the whitelist, its access to common BUS is blocked. In most recent work [21], the authors developed a defense against keyboard hijacking attacks. The main idea is people's typing pattern is different from typing pattern of such attacks and their system can utilize this difference to block the access of UWB Malware. This solution has a problem of false alarm which could be annoying for users. Also, attackers can mimic user typing patterns.

Our main contribution in this work is proposing a service based whitelisting approach which is able to assign unique identifiers to USB drives. All the previously mentioned solutions against BadUSB need to uniquely identify each device. In addition, our feature set includes interfaces the USB device asks during enumeration process and if there is a change in the interfaces, the generated ID will differ from original one and the infected device will be identified by our firewall.

7. DISCUSSION

Currently, many organizations ban their employees to use USB devices to avoid problems with this new generation of malware. Our suggestion is building a whitelist of trusted USB drives and block the rest of devices. In this case, people can continue using devices during their daily duties. This is practical because companies easily can collect trusted USB device's features and build such a whitelist.

Our approach can not guarantee to detect all types of reflashed whitelisted devices. We generate the fingerprint for each device based on several features including interfaces and services the USB device asks for. If an attacker modifies the firmware and after modification, the USB device asks for new interfaces or services (BadUSB does that), the newly generated fingerprint will be different from original one and the whitelist will block the device. But, if the attacker somehow finds a trusted device and patches the firmware without changing any of the original interfaces, the malicious device will not be detected by our system, nor would other solutions in research or industry.

Interesting fact which can be inferred from 5 is only the

top seven ranked features give us more than 95% identification accuracy. In other words, in our final whitelist creation application we just need to use seven features to build a whitelist with 95% accuracy. This result seems to be important in the cases that we do not have access to all 24 features, using just 7 of them still provides us resealable accuracy.

8. CONCLUSIONS

The only practical approach against BadUSB is creating a whitelist of trusted USB devices. In that case, all USB devices which do not belong to whitelist are blocked or restricted by our application. Creating a whitelist of USB devices is challenging. In our work, we proposed an effective approach to create whitelist of USB drives. We evaluated performance of our technique using real data collected from USB devices used by students over one year. Our technique can uniquely identify each USB device with accuracy of 98.5%. In addition, our solution will detect changes in device's primary usage and block the device if it asks for services other than original ones. We also validated the feasibility of our approach by launching a BadUSB attack from an infected device to a Windows and Linux host.

9. REFERENCES

[1] Badandroid. https://opensource.srlabs.de/projects/badusb. Accessed: 2015-09-24.

[2] Controlling device driver installation. https://technet.microsoft.com/en-us/library/cc731387(WS.10).aspx. Accessed: 2016-09-24.

[3] Deviceinstanceid. https://msdn.microsoft.com/en-us/library/windows/hardware/ff541327.aspx. Accessed: 2016-09-24.

[4] Everstrik device whitelist. http://www.everstrike.com/usbsecurity/help/device-whitelist.htm. Accessed: 2016-09-24.

[5] Gdata software ag. how to be sicher from usb attacks. https://www.gdata.at/at-usb-keyboard-guard. Accessed: 2015-09-24.

[6] How to allow authorized usb access on your network. http://www.solarwinds.com/log-event-manager/usb-access.aspx. Accessed: 2016-09-24.

[7] Ironkey secure usb devices. http://www.ironkey.com/en-US/solutions/protect-against-badusb.html. Accessed: 2016-06-03.

[8] List of usb id manufacturers. http://www.linux-usb.org/usb.ids. Accessed: 2016-09-24.

[9] Lumension end point security. http://bowmantec.com/eng/endpoint-security/. Accessed: 2016-09-24.

[10] Symantec data loss prevention. http://www.symantec.com/connect/articles/create-white-list-usb-disk-dlp-agent. Accessed: 2016-09-24.

[11] Ubuntu-gaurdian. http://ubuntuforums.org/showthread.php?t=2158605. Accessed: 2016-09-24.

[12] Usb class codes. http://www.usb.org/developers/definedclass. Accessed: 2016-09-24.

[13] Usb device registry entries. http://msdn.microsoft.com/en-us/library/windows/hardware/jj649944.aspx. Accessed: 2016-09-24.

[14] Usb guard. https://github.com/dkopecek/usbguard. Accessed: 2016-09-24.

[15] Usb guardian. http://www.ghacks.net/2010/11/07/usb-waechter-only-allow-whitelisted-usb-devices-pc-access/. Accessed: 2016-09-24.

[16] Whitelisting-and-blacklisting. http://www.gfi.com/products-and-solutions/network-security-solutions/gfi-endpointsecurity/specifications/whitelisting-and-blacklisting. Accessed: 2016-09-24.

[17] Microsoft Security Intelligence Report,Vol 20. Technical report, Microsoft Inc., Dec 2015.

[18] J. Kang and D. Lee. Advanced white list approach for preventing access to phishing sites. In *Convergence Information Technology, 2007. International Conference on*, pages 491–496. IEEE, 2007.

[19] L. Li, E. Berki, M. Helenius, and S. Ovaska. Towards a contingency approach with whitelist-and blacklist-based anti-phishing applications: what do usability tests indicate? *Behaviour & Information Technology*, 33(11):1136–1147, 2014.

[20] M. Neugschwandtner, A. Beitler, and A. Kurmus. A transparent defense against usb eavesdropping attacks. In *Proceedings of the 9th European Workshop on System Security*, page 6. ACM, 2016.

[21] S. Neuner, A. G. Voyiatzis, S. Fotopoulos, C. Mulliner, and E. R. Weippl. Usblock: Blocking usb-based keypress injection attacks. In F. Kerschbaum and S. Paraboschi, editors, *Data and Applications Security and Privacy XXXII*, pages 278–295, Cham, 2018. Springer International Publishing.

[22] K. Nohl, S. Kribler, and J. Lell. Badusb, oa accessories that turn evil. In *BlackHat Conference Proceedings*, pages 84–89. Security Research Lab, Auguest 2014.

[23] R. Rao and S. Ali. A computer vision technique to detect phishing attacks. In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*, pages 596–601, April 2015.

[24] S. Sikka, U. Srivastva, and R. Sharma. A review of detection of usb malware. *International Journal of Engineering Science*, 14283, 2017.

[25] D. T. Sullivan. Survey of malware threats and recommendations to improve cybersecurity for industrial control systems version 1.0. Technical report, DTIC Document, 2015.

[26] D. J. Tian, A. Bates, and K. Butler. Defending against malicious usb firmware with goodusb. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 261–270. ACM, 2015.

[27] B. Yang, D. Feng, Y. Qin, Y. Zhang, and W. Wang. Tmsui: A trust management scheme of usb storage devices for industrial control systems. *IACR Cryptology ePrint Archive*, 2015:22, 2015.

[28] M. Yoon. Using whitelisting to mitigate ddos attacks on critical internet sites. *Communications Magazine, IEEE*, 48(7):110–115, 2010.