Plug and Prey: Malicious USB Devices

Adrian Crenshaw
Presented at Shmoocon 2011

Abstract

This paper is meant as an overview of malicious USB devices. The paper will first lay out a proposed set of categories for malicious USB devices, how these categories function, how they differ, and how they can be used by an attacker. Next the paper will offer information on how these threats can be technically detected and mitigated, as well as human practices that can help alleviate the threats posed by malicious USB devices.

Sections:

1. Introduction

While a fair amount of research has gone into blocking malicious software (viruses, worms, trojans, spyware, etc.), comparatively less time has been spent researching malicious hardware devices. There are many examples of malicious hardware, to name just a few: backdoored routers, surreptitiously installed hosts that act as pivots on a network, PS/2 key loggers, etc. The topic of malicious hardware can be pretty broad, so this paper will be concentrating specifically on malicious USB devices. USB devices are of special interest as they often require less user interaction to install on a system than other types of hardware peripheral (PCI cards for example) meaning less attention may be paid to what tasks they are doing under the user's nose. While modern operating systems have ways to help mitigate the threats, little seems to be done by current security systems to thwart malicious USB devices. Purely software based attacks may be initiated inexpensively since once the software is developed it is practically free to replicate the attack against further targets. Since there is a minor cost to each attack because of hardware expense, we imagine that hardware attack vectors will be mostly targeted against those with data valuable enough to warrant the expense. The purpose of this

paper is to inform the reader about different classes of malicious USB devices, what can be done to protect systems from such hardware, make recommendations as to best practices to secure environments, and to increase awareness of malicious USB devices in general.

2. Background

It might be helpful to start off with a list of the generalized categories that malicious USB devices could fall into. While these categories may be subdivided further, the four family groups this paper will define are: USB Mass Storage devices containing malware (malicious software), U3 thumb drives with "evil" autorun payloads, hardware key loggers and “Programmable HID USB Keyboard Dongle” devices. What follows are descriptions of each category, how they differ from each other, as well as references to their use if available.

2.1 USB mass storage containing malware

The USB mass-storage device class is a standard way to present digital storage to a host operating system. Since it is a well-accepted standard, each individual device type does not need to ship with its own drivers for presenting the onboard storage to the user. The USB device should just work as long as the vendor follows the specifications of the USB mass-storage device class, and the host the storage is being plugged into understands these specifications. Many devices utilize the USB mass-storage device class standard: flash drives, card readers, digital cameras, MP3/media players, digital picture frames and a plethora of other items.

The greatest security risk of USB mass-storage devices, besides perhaps data leakage caused by lost devices or insiders stealing sensitive files, is as another vector for more traditional malware to spread. In other words, the device is not malicious in and of itself, but it can contain files (malware) that are malicious in nature. Sometimes lack of proper security and quality control on the manufacturer’s end can lead to malware being on a USB storage device, even if the end user has yet to open the packaging.

In the recent past USB devices have shipped from manufacturers with malware already on their internal storage. To give but a few notable examples:

Mariposa botnet client on the Vodafone (Bustamante, 2010)

Malware shipped on Apple Video iPods (SophosLabs, 2006)

Digital Photo Frames and Other Gadgets Infected with Malware (Zetter, 2008)

Malware shipped on devices from the factory may be installed onto a system via manual or automatic means. In the case of manual install, the onboard storage may come with extra software such as a backup application that the user may choose to install manually, or extra drivers if the USB peripheral is a composite device having more than just storage functionality. This extra software could contain malware. In the case of automatic installation, the device may utilize an autorun.inf file intended to make the installation of the extra software more user friendly. If autorun is enabled for the device type, the malware may be installed without user

interaction. The malware need not rely on autorun being enabled however; there are exploits that may allow the malware on the USB storage to be run merely by accessing the drive in a file browser. Two notable examples of such exploits are the WMF (Windows Metafile) vulnerability (Microsoft , 2006) and the more recent LNK (Link) file vulnerability (Leyden, 2010). Both of these attacks can lead to code being executed in the context of the current user merely by browsing the containing folder in Windows Explorer.

While these incidents highlight the need for greater quality control in the manufacturing process, we do not find any of them particularly interesting from a novelty or USB specific standpoint. Ultimately they are merely examples of malware being run from storage; little different than CDs, hard drives and the floppies of old that used to be the major vectors of malware distribution. They are also of less interest to us because they are not necessarily of deliberate intent, at least from the perspective of the company that shipped them (items like Sony's DRM rootkits are another story) (Schneier, 2005). One key aspect that is of incident response interest to us is the capability for the malware payloads to be automatically run upon the insertion of the device. This has also been seen before, back in the bygone era when people would leave a floppy in their drive by accident that had a boot sector virus on it, which then installed itself on the next disk that was inserted after booting. The less user interaction that it takes for a piece of malware to run, the better its chance of doing its intended task while remaining unnoticed by the user. This leads into the next category of malicious USB hardware we wish to cover.

2.2 U3 thumb drives with "evil" autorun payloads

This could technically be lumped in with the "USB mass storage containing malware" category. It has many similarities with the first category covered; the main reason we break it out into its own category is the deliberate intent aspect. While USB mass storage containing malware may be inadvertently caused by an accidental infection at the factory, U3 Thumb drives with "evil" autorun payloads are generally created intentionally.

First, let's give a little background information on how a U3 flash drive differs from a standard flash drive. Essentially, a U3 thumb drive has a special partition that is seen as a CD-ROM device when the thumb drive is enumerated by the host. CD-ROMs can utilize an autorun feature which may be set to automatically execute an application or script off of the device, depending on the operating system (and OS version) being used. In Microsoft Windows, the file autorun.inf on the U3 drive's "CD-ROM" section is consulted to decide what to execute, with some subtle variations between the different versions of Windows. The original intent of the U3 thumb drive was to make it easier for the user to access applications on the drive by automatically popping up a menu when the device is inserted that allows the user to select portable applications to run off of the U3 thumb drive. This intent has, of course, been subverted for other tasks as well.

The community founded around the HAK5 video podcast has done a lot with the U3 autorun concept (HAK5 Wiki). The essential idea of HAK5's USB Switchblade/Hacksaw is to automatically run applications and scripts upon the insertion of the U3 flash drive, with varying results. One of the most popular tasks to have the U3 USB drive carry out is to collect locally

stored passwords and password hashes, then save this sensitive information to the flash drive's storage. While password collection seems to be the most popular task, many payloads have been devised, and not all of them are even intended to help compromise a system. For example, Russell Butturini's Incident Response payload is designed to quickly grab useful information from a system before it is taken offline, aiding the investigator in figuring out what happened to the box while minimizing the amount of time the box has to stay live on the network (Butturini). The key advantage of the U3 USB flash drive from the attacker's perspective is that they can be automated, thus making the collection of the data faster and less likely for the owner of the system to notice what the attacker is doing.

U3 thumb drives have also been used in another capacity: as Trojans left behind to be found by unsuspecting users. Few users can resist the lure of a free flash drive. There have been cases where a penetration tester left U3 thumb drives lying around where people in a certain organization could find them, and was successful in compromising several systems when users from the facility picked them up and started using them (Darkreading, 2006) (Johansson, 2008).

Steve Stasiukonis of Secure Network Technologies Inc. conducted an experiment during one of his penetration tests against a credit union. His team collected vendor imprinted flash drives that they had received (presumably as promotional items) and created a custom Trojan to install on them. The custom Trojan they designed would collect logins, passwords and machine specific information, and then email these details back to the team. The trojaned drives were then spread around the credit union's parking lot and other locations where the employees might find them. Stasiukonis reports that of the twenty trojaned flash drives that they left behind, fifteen were found by employees and all fifteen were plugged into company computers. Information they obtained via this vector was then used as a stepping stone to further compromise the credit union's systems.

On Windows Vista and newer Microsoft Operating Systems the autorun/autoplay capability is not nearly as silent, nor automatic, as it once was. Still, it can lead to issues caused by unwitting users. Microsoft has published details on properly configuring autorun for Windows 200/XP/2003 (Microsoft, 2010) and details on changes in their newer Operating Systems like Windows Vista/2008/7 (Microsoft, 2010).

2.3 Why this paper will focus on the last two categories

Now that we have covered the basics of what "USB Mass Storage devices containing malware" and "U3 thumb drives with evil autorun payloads" are, let's discuss why they will not be the primary focus of this paper. Both of these first two categories are basically ways to install malware. Anti-malware is a fairly well covered topic, and these storage based classes of USB devices are just two more vectors. Anti-malware packages from major vendors are already focusing on these issues, albeit with mixed results (especially where custom coded malware is concerned). The autorun capabilities of different types of removable media make them somewhat different than malware that spreads over a network, or via a user deliberately choosing to run a binary, but disabling or reconfiguring autorun policies is already covered pretty well in other documents.

The last two categories of malicious USB hardware are significantly different. While the first two are merely a means to deliver malware, “hardware key loggers” and “Programmable HIDs” can be malicious in and of themselves, without ever having to install any malware onto a system. Disabling autorun does not help to mitigate their risks since they do not use that functionality to work, and since they don’t have to install any software on a system to “do their dirty deeds”, anti-malware engines have little to do with them. These features make the categories of “Hardware key loggers” and “Programmable HIDs” fundamentally different from those that are strictly storage based attacks.

2.4 Hardware key loggers

Hardware key loggers are fairly simple devices conceptually. Essentially they are installed between the keyboard and the computer, and then log all of the keystrokes that they intercept to their onboard flash memory. A snooper can then come along later to pick up the key logger and extract the captured data (passwords, documents, activity, etc.).

Installing a hardware key logger normally requires physical access to the target system; though there are ways to socially engineer users into installing them. For example, an attacker could give the key logger to the victim and claim it is a copy protection dongle for a trial version of a software package they wish to use. When they find out the software has expired anyway, the attacker tells the victim that the “licensing dongle” must be broken and asks them to mail it back for replacement. At this point, the attacker can look at the data on the key logger. Also, some hardware key loggers are coming to market that only need the snooper to obtain direct physical access initially to install the devices. The keystrokes can then be recovered wirelessly later, though the range needed for recovery varies, so close proximity may be required (KeeLog) (Wirelesskeylogger) .

The practicality of USB hardware key loggers from the attacker’s perspective varies for many reasons:

1. The cost can be high, from about $60 to well over $300. How many attackers would wish to leave behind such an expensive piece of equipment that they may not be able to collect later?
2. Physical access to the system is required for a short time to install, and possibly to retrieve. So far the wireless key logger the author has tested has been less than reliable.
3. USB is a tougher spec to implement than PS/2. Some keyboard/PC combinations simply fail to function, or function with odd side effects.

These issues make it less likely that hardware key loggers will be deployed except against places where there is a strong financial (corporate), political (government) or personal (spouse/children) motive.

Detection and mitigation

As far as detection of USB hardware key loggers, this varies greatly from brand to brand. Some loggers, such as the KeyCarbon, appear as a USB hub device. Unfortunately, the KeyCarbon's USB vendor id is 0451 and its product id is 2046, both of which belong to a Texas Instrument hub. These mundane identifiers make it pretty hard to filter reliably by USB IDs, as will be described later in this paper. Others brands of USB key logger we have tested seem to be completely passive from the host system's perspective. In other words, they do not show up as any sort of device under "Device Manager" in Windows or under the /dev/ or /sys/ directories in Linux. Some do however cause problems that a user may notice when a keyboard with a built-in USB hub is used. If devices are plugged into the keyboard's hub while the hardware key logger is installed the extra devices may not function at all, or function at a slower speed (USB 1.1 as oppose to 2.0 for example). This speed issue was noticed with the KeyCarbon and KeeLog units that were tested.

For further information on the varying brands of hardware key logger the author has tested, please see the links provided in Works Cited (Crenshaw, Irongeek's Key logger Research, 2007).

Essentially, detection of a USB key logger comes down to three points:

1. Physically observe that one is connected to a host. While internal hardware key loggers do exist, most are external.
2. Take note of the odd behaviors of other devices plugged in line with the keyboard. Some keyboards also have built in hubs so that other devices such as a thumb drive or an MP3 player may be conveniently plugged into them. Having a USB key logger in line with these devices may cause them to function poorly, or not at all.
3. Take note of odd vendor and product IDs in Device Manager. One key logger we have encountered identified itself as an Apple Keyboard, which might be an anomaly if found on a commodity Windows or Linux PC. Unfortunately for someone trying to detect it, this "Apple Keyboard" only appeared when the logger was in keystroke recovery mode. If the attacker takes the device back to their own machine for keystroke recovery it will never appear on the compromised system. Black listing USB vendor or product ID may not be such a great countermeasure, as such devices either have a very generic identifier ("TI hub" for example) or no identifiers left behind at all.

More details on blocking USB devices from being installed will be covered in the Windows Local Security Policy and Linux UDEV sections.

2.5 Programmable HID USB Keyboard Dongle Devices

This type of malicious USB hardware is somewhat different from the other categories, and has only recently gained much attention in the security community. To create a

Programmable HID (Human Interface Device) a microcontroller is programmed to act as a USB HID keyboard and mouse, and set to send certain keystrokes and mouse movement under a given set of circumstances (such as time, lighting, keyboard LED states, etc.). There are many tasks these programmable USB HIDs can carry out, but some common tasks that an attacker/pen-tester may want to program one for are:

1. Add a user account to the system or the domain.
2. Run a program that sets up a backdoor.
3. Copy files to a thumb drive, or upload them to a site that the attacker controls.
4. Go to a website that the victim has a session cookie for, and initiate some sort of transaction (vaguely like a cross site request forgery attack, but hardware based).

The advantage of a USB HID for these sorts of functions over a U3 flash drive is that with a USB HID it does not matter if autorun is disabled or not. By default, most operating systems seem to automatically install and configure USB HIDs as soon as they are inserted, regardless of the privilege level of the current user. There are however other methods to keep rogue HIDs from being installed, which we will go into later in this article.

So far, to the author's knowledge, there has been little real world deployment of programmable HIDs besides their use as pranks. However, four different presentations at DefCon 18 and surrounding conferences (Black Hat USA 2010 and B-Sides Las Vegas 2010) involved using programmable HIDs, so we might expect to see more use of this class of device soon. The highlights from these conference presentations are as follows:

Dave Kennedy used a programmable HID (also known as a PHUKD, short for Programmable HID USB Keyboard/Mouse Dongle) in his "SET 0.6 Release with Special PHUKD Key" talk at B-Sides Las Vegas 2010. Since Mr. Kennedy integrated the PHUKD with his Social Engineering Toolkit (SET) we can probably anticipate others trying out this vector of attack. Dave Kennedy and Josh Kelly also used the device in their talk "Powershell...omfg" at Defcon 18 and Black Hat Las Vegas 2010 the same week. In this talk they demonstrated using Metasploit type functionality but leveraging Microsoft Powershell with the PHUKD.

Richard Rushing gave a talk called "USB - HID, The Hacking Interface Design" at Black Hat USA 2010.

Monta Elkins demonstrated a radio controlled version of the programmable HID concept in his talk "Hacking with Hardware: Introducing the Universal RF USB Keyboard Emulation Device – URFUKED". Instead of having the programmable HID wait for a timer, or for certain environmental conditions to be met, Mr. Elkins' device was triggered via an RF based remote control.

Finally, there was the author's talk "Programmable HID USB Keystroke Dongle: Using the Teensy as a Pen Testing Device". It covered the basics of constructing a programmable HID. Also covered were potential ways to hide the device in other hardware or cubicle toys, and socially engineering a target into installing the device.

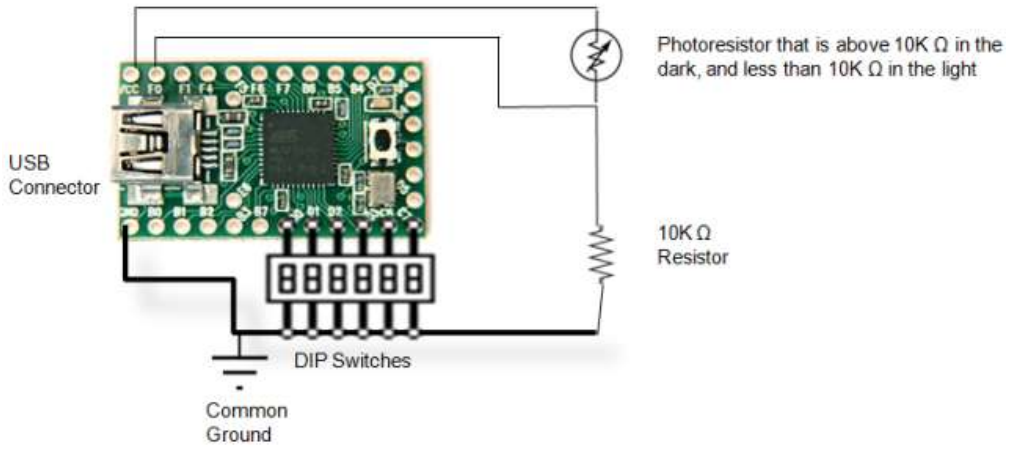
The author's version of the device is relatively easy to construct and program. The core chip is the Teensy 2.0 or Teensy 2.0++, which are AVR based microcontroller development boards. Dave Kennedy's, Monta Elkins' and Richard Rushing's versions are based on this platform as well. The programing environment is Arduino, which is essentially a set of wrappers around C++ that make the development easier for novices to microcontrollers. The author of this paper has also developed an Arduino library that extends and simplifies the commands that come with the Teensy for sending common commands to a host operating system. The Arduino development environment makes programming tasks relatively easy. Figure 1 shows a complete program that, when combined with the PHUK library and the hardware from the schematic, will cause the Teensy to wait for lighting conditions in the room to change, then open a command prompt in Windows and add a user to the system.

Figure 1: Simple PHID Example

```
#include <phukdlib.h>
int PhotoRead;
int OldPhotoRead;
void setup(){
  delay(10000);//Give time for device to enumerate
  PhotoRead = analogRead(0);
  OldPhotoRead = PhotoRead;
};
void loop()
{
  PhotoRead = analogRead(0);
  if (abs(PhotoRead - OldPhotoRead) > 100 ) {
    CommandAtRunBarMSWIN("net user hacker l3tme1nn0wplease /add /y");
    delay(10000);
    OldPhotoRead=PhotoRead;
  }
}
```

Simpler or more complex keyboard command sets could also be constructed.

The schematic for a basic PHUKD device is relatively simple:



Not all of the components shown are necessary, for example the 10KΩ resistor and photoresistor could be removed if the ambient light sensor is not needed. The example source code in Figure 1 can be modified to use the light sensing capabilities to decide what payload to run based on how bright a room is, and can also be used as a simple motion detector. The attached dipswitches are also arbitrary, the number of positions can vary based on the options a user needs to select, or could be omitted completely if the user does not need to make any in-the-field adjustments to the PHUKD's intended task. The PHUKD library, available from the project's website, has additional source code that demonstrates other payloads. Please see the project's website for more details on the hardware and programming the device. (Crenshaw, Programmable HID USB Keystroke Dongle: Using the Teensy as a pen testing device, 2010).

Further research is intended for this device class. Some of the author's key goals are to find longer range ways to control the device over RF, and to integrate a key logger into the same package so that the programmable HID could react better to user activity as well as store user names and password for later use.

Detection and mitigation

Programmable USB HIDs must appear as devices on a system's Universal Serial Bus since the operating system has to load the drivers for them to function, albeit drivers that are already part of the OS and are installed automatically by default when the USB device is inserted. By their very nature programmable HIDs cannot be completely passive devices like many hardware key loggers can. There are options available in Windows and Linux for restricting the automatic installations of devices, which we will go into in later sections. Blacklisting USB vendor and product IDs is not a reliable option since the creator of the programmable HID device can change these identifiers to anything they want, including the IDs of devices that are in a target's whitelist. One possible check for these devices is to programmatically look for systems that have more than one keyboard plugged in, which would normally be an anomaly. Current versions of the PHUKD device report themselves as a keyboard and mouse as soon as they are plugged in, but there is no technical reason why future versions can't be set to only inform the system of their presence once the payload needs to be typed, and not before. This would make scanning for systems with more than one keyboard somewhat pointless. The only current reliable solution for finding the presence of such devices is close physical inspection and caution when it comes to attaching USB devices to your system. Guidelines are given later in the paper on how to disable the automatic installation of USB HID devices, though this solution may not be very convenient for some users.

3. Locking down Windows and Linux against Malicious USB devices

The following sections will cover how USB based attack vectors may be mitigated in both Windows and Linux. Other platforms are not covered because of the author's lack of access to the needed environments.

3.1 Locking down Windows Vista and Windows 7 against Malicious USB devices

A fair amount has been written already about locking down a Microsoft Windows system to protect it against undesired USB flash drive usage. If system owners want to keep data from leaving their network via removable storage there's a simple registry entry that can be tweaked:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\StorageDevicePolicies\WriteProtect
```

Setting this registry entry to 1 will cause USB storage devices to be read only. A simple tool the author wrote awhile back can also be used to manipulate this registry key (Crenshaw, Thumbscrew: Software USB Write Blocker, 2007). This, of course, is not a perfect solution; there are ways around it if the attacker has sufficient time along with their physical access, but this method is more flexible and less destructive to the hardware than the epoxy solution some choose.

It should be noted that the tool above was meant more for forensics use (though we make no guarantee that it is forensically sound). A security professional may be worried about more than just data leaking out of their systems on removable storage; they may also worry about U3 thumb drives with undesired Autorun payloads. In the case of Autorun/Autoplay concerns, Microsoft has a Knowledge Base article with a great amount of detail on disabling Autorun for selected device types (Microsoft, 2009).

What this section of the paper will concentrate on is stopping other classes of possibly malicious USB devices, especially the PHUKD (Programmable HID USB Keyboard/Mouse Dongle) mentioned previously. One of the advantages of the PHUKD is that human interface devices (HIDs) like mice and keyboards don't require administrative privileges to be installed and function, at least by default. Another advantage the PHUKD has is that many organizations are beginning to lock down Autorun on their systems to prevent malware like Conficker from spreading via that particular vector, and to prevent tools like the Hak5 U3 Hacksaw from functioning. However, since a PHUKD is a USB HID, turning off Autorun has no effect on it. There are however other Windows 7/Vista settings that can be tweaked to disable arbitrary USB devices.

While we did most of our testing of the following Windows Vista/7 security options using a PHUKD device, they should also prove useful in blocking U3 thumb drives, WiFi dongles (think inadvertent rogue access points), non-passive key loggers and other devices that could be attached to a system. Also, these security options can be applied to restrict other types of hardware, not just USB, though USB peripherals are what we will concentrate on in this section.

Shortly we will be covering Windows 7/Vista Group Policy/Registry tweaks that can be applied to block the automatic install of USB devices, but first there is a tool you may want to download to ease your experimentation. Nirsoft's USBDeview was of great use to us during this research, and much more suited to the tasks we needed to accomplish than Windows Device Manager.

http://www.nirsoft.net/utils/usb_devices_view.html

A few of the more useful features of USBDeview include:

1. View Vendor ID, Product ID, Device Class, Serial Number and etc. all from one line of output.
2. Uninstall devices, even if they are not currently connected to the system.
3. Jump straight to the registry keys related to a USB device.
4. Export a list of installed USB devices to a text file.

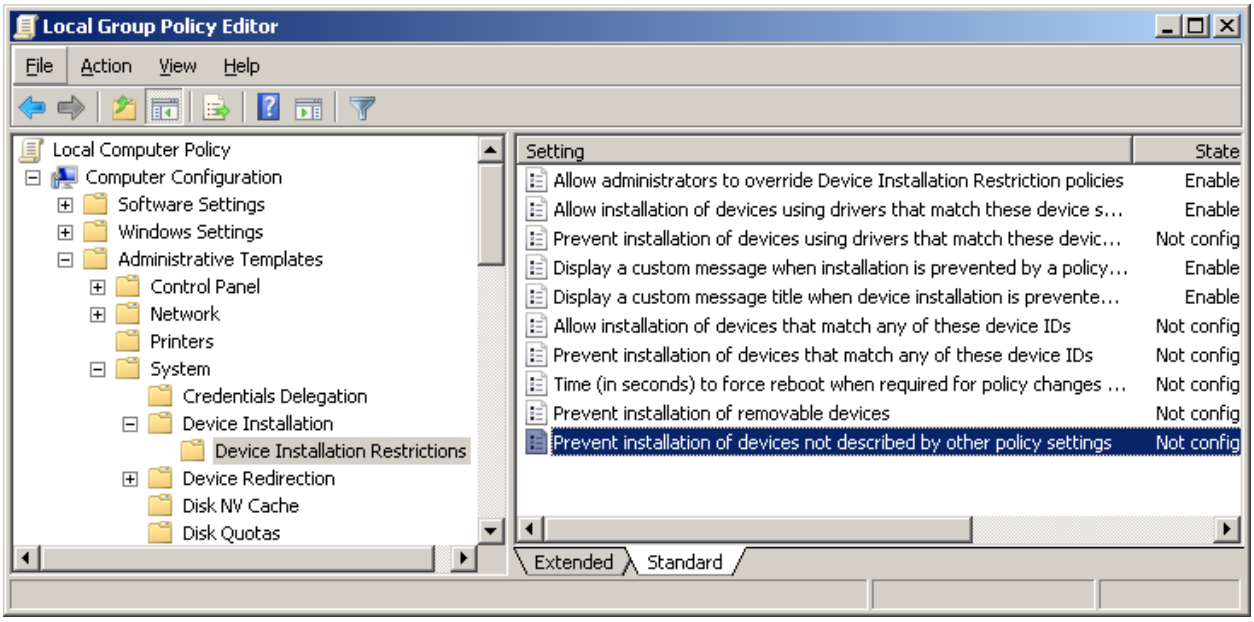
Along with USBDeview it may be useful for you to be able to quickly open the MMC plugins we will be using in this article: Device Manager and Local Group Policy Editor. To jump directly to these MMC plugins: Enter the command “devmgmt.msc” to bring up the Device Manager, or “gpedit.msc” to bring up the Local Group Policy Editor. These commands may be entered via the “Search programs and files” bar, the Run bar or via the command console (cmd.exe/ powershell.exe). Putting shortcuts to them on the Desktop is also an option, of course.

Now that we have the needed tools, let’s cover the Device Installation Restriction options available in Windows 7/Vista. For each entry the following information will be given:

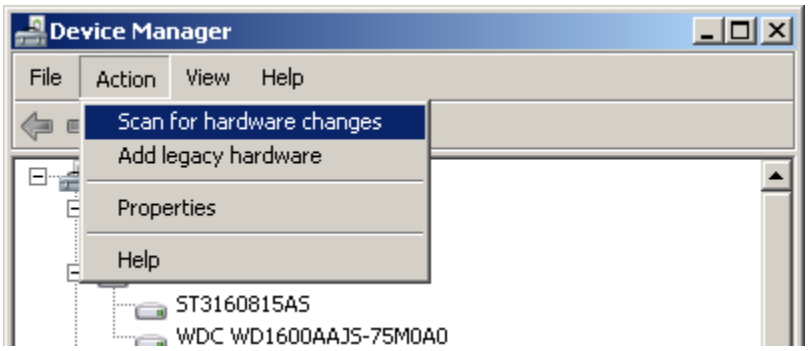
1. The setting’s name.
2. A quote of Microsoft’s description of the setting as seen in the Group Policy Editor.
3. Our notes from testing where we will try to clarify the use of the settings and certain “gotchas” you may encounter while using them.
4. The registry keys and values that are changed when the option is enabled.
   Note: This may make it possible for administrators to create their own scripts and interfaces for manipulating these security options. However, keep in mind that when these Policy Objects are created they receive their own IDs ({55803F47-01A6-4A85-89CE-74357A125D17} in many of our examples). The exact path seen for the manipulated registry keys is based on what was changed on our test system, and is very unlikely to be the same on other systems.

Now let’s take a look at some of the Group Policy Options (GPO) Windows 7/Vista provides for restricting hardware installation. To bring up the list of Device Installation Restriction options, use the command “gpedit.msc” as covered earlier, and navigate to:

Computer Configuration->Administrative Templates->System->Device Installation->Device Installation Restrictions



If at any point you have problems getting hardware to work because of changing these setting, set all of these GPO options to “Not Configured” then go into Device Manager and do an “Action->Scan for Hardware Changes” from the menu bar.



List of Windows 7/Vista hardware installation policy options:

Allow administrators to override Device Installation Restriction policies

Allow installation of devices using drivers that match these device setup classes

Prevent installation of devices using drivers that match these device setup classes

Display a custom message when installation is prevented by a policy setting

Display a custom message title when device installation is prevented by a policy setting

Allow installation of devices that match any of these device IDs

Prevent installation of devices that match any of these device IDs

Time (in seconds) to force reboot when required for policy changes to take effect

Prevent installation of removable devices

Prevent installation of devices not described by other policy settings

Allow administrators to override Device Installation Restriction policies

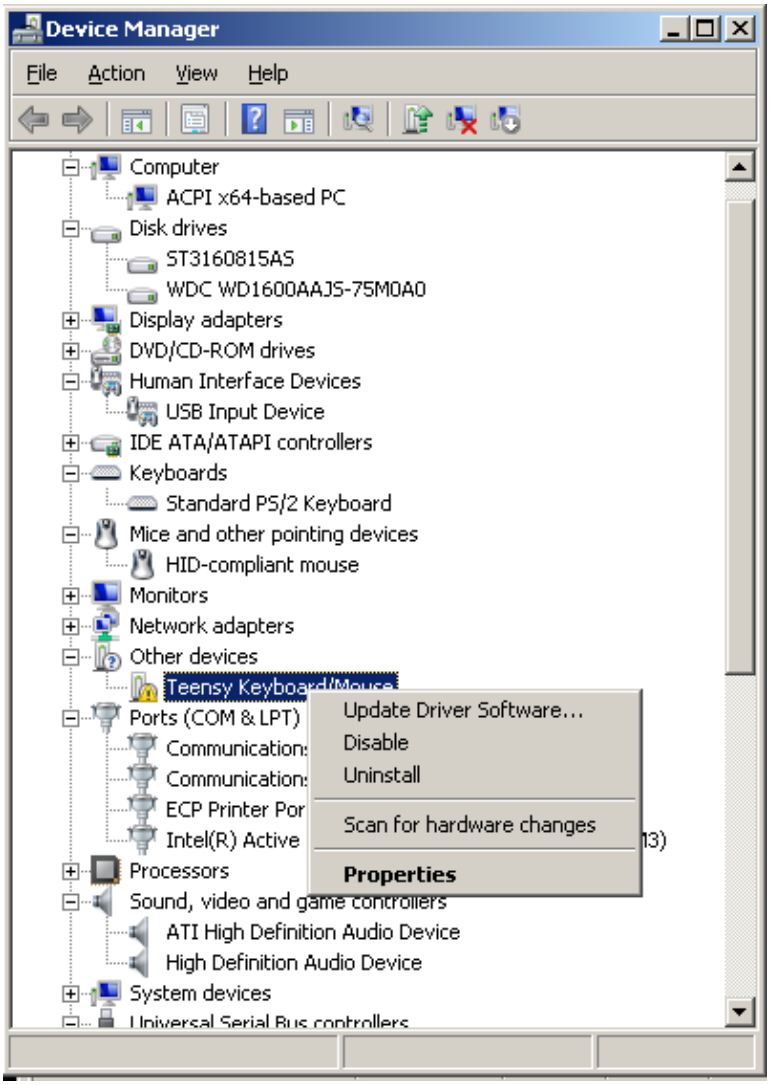
Microsoft's Description: This policy setting allows you to determine whether members of the Administrators group can install and update the drivers for any device, regardless of other policy settings.

If you enable this policy setting, members of the Administrators group can use the Add Hardware wizard or the Update Driver wizard to install and update the drivers for any device. If you enable this policy setting on a remote desktop server, the policy setting affects redirection of the specified devices from a remote desktop client to the remote desktop server.

If you disable or do not configure this policy setting, members of the Administrators group are subject to all policy settings that restrict device installation.

Author's Notes:

As should be obvious, this setting will have no effect unless you set one of the "Prevent" options listed below. If one of the policy settings prevents a USB device from being installed, and the "Allow administrators to override Device Installation Restriction policies " option is set, an admin can go into device manager to install the device. Doing a simple "Action->Scan for Hardware Changes" will not work however. An administrative user will have to go into Device Manager, find the device that was prevented from automatically installing, then right click it and choose "Update Driver Software…" This should force the installation of the device.



Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions]
```

Value if enabled:

```
"AllowAdminInstall"=dword:00000001
```

Allow installation of devices using drivers that match these device setup classes

Microsoft's Description: This policy setting allows you to specify a list of device setup class globally unique identifiers (GUIDs) for device drivers that Windows is allowed to install. Use this policy setting only when the "Prevent installation of devices not described by other policy settings" policy setting is enabled. Other policy settings that prevent device installation take precedence over this one.

If you enable this policy setting, Windows is allowed to install or update device drivers whose device setup class GUIDs appear in the list you create, unless another policy setting specifically prevents installation (for example, the "Prevent installation of devices that match these device IDs" policy setting, the "Prevent installation of devices for these device classes" policy setting, or the "Prevent installation of removable devices" policy setting). If you enable this policy setting on a remote desktop server, the policy setting affects redirection of the specified devices from a remote desktop client to the remote desktop server.

If you disable or do not configure this policy setting, and no other policy setting describes the device, the "Prevent installation of devices not described by other policy settings" policy setting determines whether the device can be installed.

Author's Notes:

This option allows you to create a whitelist of devices that may be installed. As noted above, this setting does nothing unless "Prevent installation of devices not described by other policy settings" is set. Also, "Prevent" policies override all of the "Allow" policies except for "Allow administrators to override Device Installation Restriction policies". In other words, if you set the "Prevent installation of devices using drivers that match these device setup classes" policy to deny the installation of the "USB Input Device Class", but also set "Allow installation of devices using drivers that match these device setup classes" to allow the installation of the "USB Input Device Class", then the Prevent policy will take precedence.

While this setting allows you to create a whitelist, it is rather painful to do so as you have to allow all of the associated Device Classes for a given device. For example, we had to enable the following to allow the Teensy based PHUKD device to be installed:

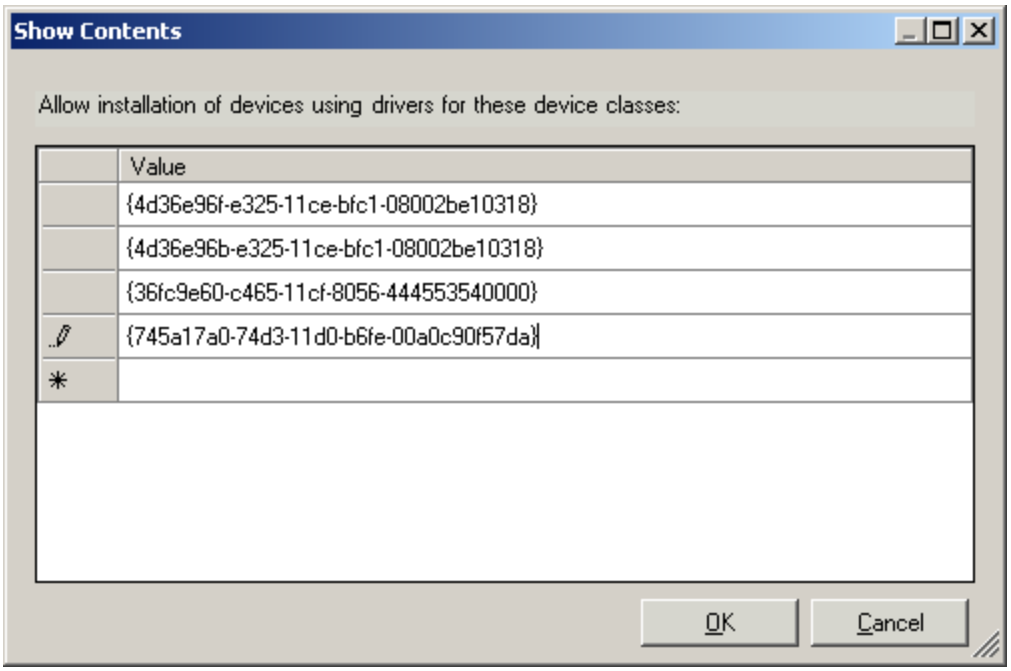
{4d36e96f-e325-11ce-bfc1-08002be10318} (HID-Compliant Mouse)

{4d36e96b-e325-11ce-bfc1-08002be10318} (HID Keyboard Device)

{36fc9e60-c465-11cf-8056-444553540000} (Composite Device)

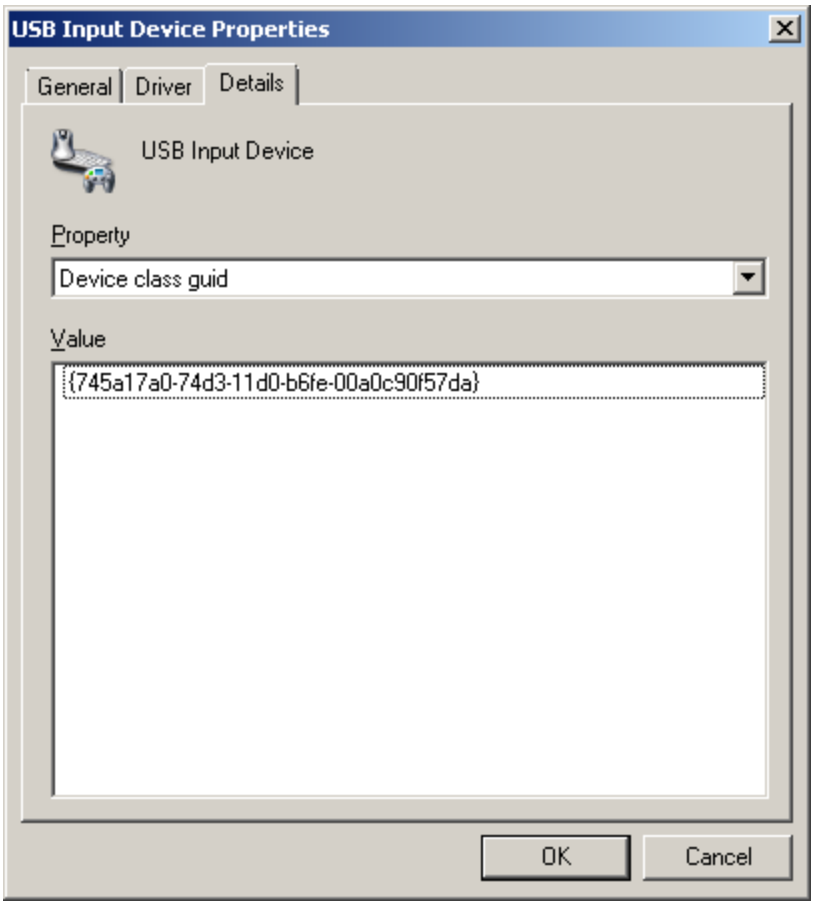
{745a17a0-74d3-11d0-b6fe-00a0c90f57da} (USB Input Device Class)

Observe this screenshot to see the entry format that must be used:



To find the needed device setup classes we followed these steps:

1. We had to let the device be installed on a box. The GUID properties for the device did not seem to be visible unless it was installed.
2. We then found the device, and all its related devices, in Device Manager.
3. We brought up the properties of each device, went to the details tab, then copied the value from the “Device class guid” property.
4. After we collected all of the “Device class guid” properties, we enabled the “Allow installation of devices using drivers that match these device setup classes” setting and added the class ids to the list.



As you can tell from the procedures listed above, using the "Allow installation of devices using drivers that match these device setup classes" option is not very admin friendly. For assistance please see the list of device GUIDs provided at the end of this section.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions]
```

Value if enabled:

```
"AllowDeviceClasses"=dword:00000001
```

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
```

```
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions\AllowDeviceClasses]
```

Values (where xxx is the Device Class):

```
"**delvals."=""

"1"="xxx"
```

Numbering should continue from 1.

Prevent installation of devices using drivers that match these device setup classes

Microsoft's Description: This policy setting allows you to specify a list of device setup class globally unique identifiers (GUIDs) for device drivers that Windows is prevented from installing. This policy setting takes precedence over any other policy setting that allows Windows to install a device.

If you enable this policy setting, Windows is prevented from installing or updating device drivers whose device setup class GUIDs appear in the list you create. If you enable this policy setting on a remote desktop server, the policy setting affects redirection of the specified devices from a remote desktop client to the remote desktop server.

If you disable or do not configure this policy setting, Windows can install and update devices as allowed or prevented by other policy settings.

Author's Notes:

"Prevent installation of devices using drivers that match these device setup classes" is pretty much the mirror opposite of "Allow installation of devices using drivers that match these device setup classes". Device setup classes are collected the same way. We imagine this setting would be useful if you know a specific class of devices you wish to block, such as USB WiFi adapters for example.

With the "Prevent" policy options, hardware that is already installed is normally ignored and stays functional even after the policy is applied. If you want to make the changes retroactive for previously installed hardware, choose the "Also apply to matching devices that are already installed" check box (or set the registry value DenyDeviceClassesRetroactive to 1). For assistance please see the list of device GUIDs provided at the end of this section.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions]
```

Values if enabled:

```
"DenyDeviceClasses"=dword:00000001

"DenyDeviceClassesRetroactive"=dword:00000000
```

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions\DenyDeviceClasses]
```

Values:

```
"**delvals."=" "

"1"="xxx"
```

Where “xxx” is a GUID. Numbering should continue from 1.

Display a custom message when installation is prevented by a policy setting

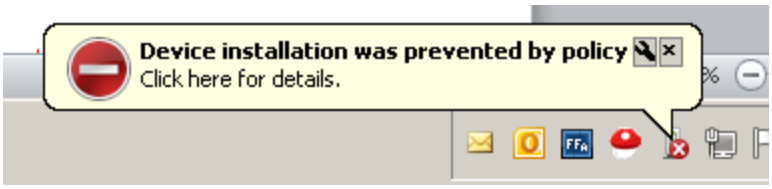
Microsoft’s Description: This policy setting allows you to display a custom message to users in the notification balloon when a device installation is attempted and a policy setting prevents the installation.

If you enable this policy setting, Windows displays the text you type in the Detail Text box when a policy setting prevents device installation.

If you disable or do not configure this policy setting, Windows displays a default message when a policy setting prevents device installation.

Author’s Notes:

This option is fairly self-explanatory. Instead of giving the default message (“Click here for details”), you can choose to give a customized message whenever a device fails to install because of policy settings.



One annoyance we've found is that these warning popups only appear the first time you try to install a given piece of hardware, so they are easy for a user to miss and not realize why their USB device is not working.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions\DeniedPolicy]
```

Value (where "message" is the message to be shown):

```
"DetailText"="message"
```

Display a custom message title when device installation is prevented by a policy setting

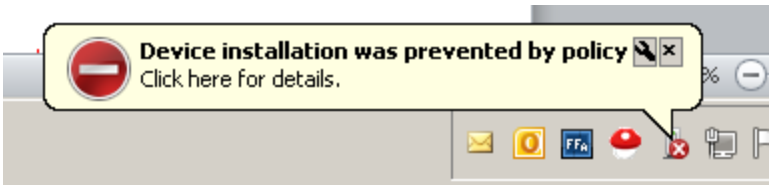
Microsoft's Description: This policy setting allows you to display a custom message title in the notification balloon when a device installation is attempted and a policy setting prevents the installation.

If you enable this policy setting, Windows displays the text you type in the Main Text box as the title text of the notification balloon when a policy setting prevents device installation.

If you disable or do not configure this policy setting, Windows displays a default title in the notification balloon when a policy setting prevents device installation.

Author's Notes:

Similar to "Display a custom message when installation is prevented by a policy setting", except it lets you set a custom title to replace the default title of "Device installation was prevented by policy".



One annoyance we've found is that these warning popups only appear the first time you try to install a given piece of hardware, so they are easy for a user to miss and not realize why their USB device is not working.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions\DeniedPolicy]
```

Value (Where "title" is the title to show):

```
"SimpleText"="title"
```

Allow installation of devices that match any of these device IDs

Microsoft's Description: This policy setting allows you to specify a list of Plug and Play hardware IDs and compatible IDs for devices that Windows is allowed to install. Use this policy setting only when the "Prevent installation of devices not described by other policy settings" policy setting is enabled. Other policy settings that prevent device installation take precedence over this one.

If you enable this policy setting, Windows is allowed to install or update any device whose Plug and Play hardware ID or compatible ID appears in the list you create, unless another policy setting specifically prevents that installation (for example, the "Prevent installation of devices that match any of these device IDs" policy setting, the "Prevent installation of devices for these device classes" policy setting, or the "Prevent installation of removable devices" policy setting). If you enable this policy setting on a remote desktop server, the policy setting affects redirection of the specified devices from a remote desktop client to the remote desktop server.

If you disable or do not configure this policy setting, and no other policy setting describes the device, the "Prevent installation of devices not described by other policy settings" policy setting determines whether the device can be installed.

Author's Notes:

This GPO option allows for another way to whitelist devices. Much like the options that "Allow" or "Prevent" installation of a device based on its device setup class, the "Allow installation of devices that match any of these device IDs" works via a list of Plug and Play hardware IDs or compatible IDs. These IDs can be somewhat easier to collect than device setup classes since the device does not have to be successfully installed first to collect the hardware IDs or compatible IDs.

Hardware IDs are meant to be rather specific to the device. They are used for finding the correct device driver to load to make the hardware functional. For example, we set the Teensy based PHUKD device to have a vendor ID of 1313 and a Product ID of 0123. This means one of its hardware IDs is:

USB\VID_1313&PID_0123

But it is also a composite device, and the parent of other devices that will appear in the Device Manager, for example:

HID\VID_1313&PID_0123&REV_0100&MI_01
HID\VID_1313&PID_0123&MI_01
HID_DEVICE_SYSTEM_MOUSE
HID_DEVICE_UP:0001_U:0002
HID_DEVICE

As such, just whitelisting the hardware ID USB\VID_1313&PID_0123 would not be enough to allow the device to completely install. As a matter of fact, it would be possible to have some of the functions of a composite device work, and have others denied because they do not have their corresponding hardware ID whitelisted. For example, the mouse part might work, but the keyboard part of the Teensy HID might not.

While hardware IDs are meant to be fairly specific to a given piece of hardware, compatible IDs are a fall back for when more specific drivers can't be found that support the listed hardware IDs. Compatible IDs are more general in other words.

To collect compatible or hardware IDs for your whitelist do the following:

1. Plug in the device.
2. Find the device, and all its related devices, in Device Manager. If the device is currently prevented from installing because of a GPO setting you may only see one device with an exclamation mark. After we finish with steps 2 through 4 on a composite device, we may have to go through them again for each child device.

3. Bring up the properties of each device, go to the details tab, and then copy a value from the “Hardware Ids” or “Compatible Ids” property.
4. After collecting all of the “Hardware Ids” or “Compatible Ids” properties, enable the “Allow installation of devices that match any of these device IDs” setting and add needed IDs to the list.

For the Teensy programmable HID device to work, I had to add the following IDs:

USB\VID_1313&PID_0123
USB\COMPOSITE
HID_DEVICE
USB\Class_03

A few other notes: Remember that “Prevent” overrides “Allow” in general, so if an ID is in both an “Allow” and a “Prevent” policy, the “Prevent” policy will generally take precedence. Also, if “Prevent installation of removable devices” is enabled, and the device is removable, it will be denied installation even if its IDs are in a whitelist. “Allow administrators to override Device Installation Restriction policies” still overrides “Prevent installation of removable devices” however.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions]
```

Value if enabled:

```
"AllowDeviceIDs"=dword:00000001
```

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions\AllowDeviceIDs]
```

Values:

```
"**delvals."=" "

"1"="xxx"
```

Where “xxx” is a hardware or compatible ID. Numbering should continue from 1.

Prevent installation of devices that match any of these device IDs

Microsoft's Description: This policy setting allows you to specify a list of Plug and Play hardware IDs and compatible IDs for devices that Windows is prevented from installing. This policy setting takes precedence over any other policy setting that allows Windows to install a device.

If you enable this policy setting, Windows is prevented from installing a device whose hardware ID or compatible ID appears in the list you create. If you enable this policy setting on a remote desktop server, the policy setting affects redirection of the specified devices from a remote desktop client to the remote desktop server.

If you disable or do not configure this policy setting, devices can be installed and updated as allowed or prevented by other policy settings.

Author's Notes:

You may use the "Prevent installation of devices that match any of these device IDs" to blacklist based on hardware or compatibility IDs. Keep in mind that this sort of blacklisting of hardware IDs can be made very ineffective because of devices that allow the attacker to set any vendor or product ID they wish. For example, we set the Teensy to use 1313 as the vendor ID, and 0123 as the product. This made the base hardware ID as follows:

USB\VID_1313&PID_0123

We could have easily changed these arbitrary values to something else, or made them match some preexisting hardware's vendor and product ID. If a blacklist is to be created it may be better to use the compatibility IDs to block device types in much the same way as the "Prevent installation of devices using drivers that match these device setup classes" uses GUIDs in its block list. If you want to make the changes retroactive for previously installed hardware, choose the "Also apply to matching devices that are already installed" check box (or set the registry value DenyDeviceClassesRetroactive to 1).

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions]
```

Values if enabled:

```
"DenyDeviceIDs"=dword:00000001

"DenyDeviceIDsRetroactive"=dword:00000000
```

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions\DenyDeviceIDs]
```

Values:

```
"**delvals."=" "

"1"="xxx"
```

Where “xxx” is a hardware or compatible ID. Numbering should continue from 1.

Time (in seconds) to force reboot when required for policy changes to take effect

Microsoft’s Description: Set the amount of time (in seconds) that the system will wait to reboot in order to enforce a change in device installation restriction policies.

If you enable this setting, set the amount of seconds you want the system to wait until a reboot.

If you disable or do not configure this setting, the system will not force a reboot.

NOTE: If no reboot is forced, the device installation restriction right will not take effect until the system is restarted.

Author’s Notes:

We’ve not really tested this option. The effects of the settings we’ve made have always seemed to be instantaneous, and not requiring a reboot. As such the author did not test this option thoroughly.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions]
```

Values if enabled:

```
"ForceReboot"=dword:00000001
```

```
"RebootTime"=dword:00000078
```

Prevent installation of removable devices

Microsoft's Description: This policy setting allows you to prevent Windows from installing removable devices. A device is considered removable when the driver for the device to which it is connected indicates that the device is removable. For example, a Universal Serial Bus (USB) device is reported to be removable by the drivers for the USB hub to which the device is connected. This policy setting takes precedence over any other policy setting that allows Windows to install a device.

If you enable this policy setting, Windows is prevented from installing removable devices and existing removable devices cannot have their drivers updated. If you enable this policy setting on a remote desktop server, the policy setting affects redirection of removable devices from a remote desktop client to the remote desktop server.

If you disable or do not configure this policy setting, Windows can install and update device drivers for removable devices as allowed or prevented by other policy settings.

Author's Notes:

Since USB devices by their very nature are generally removable this is a pretty straight forward option to set. It seems to pretty much override all other device restriction settings but the "Admin Override" option. If you want to make the change retroactive for previously installed hardware, choose the "Also apply to matching devices that are already installed" check box (or set the registry value DenyDeviceClassesRetroactive to 1).

Registry Equivalent:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group
Policy Objects\{55803F47-01A6-4A85-89CE-
74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall
\Restrictions]
```

Value if enabled:

```
"DenyRemovableDevices"=dword:00000001
```

Prevent installation of devices not described by other policy settings

Microsoft's Description: This policy setting allows you to prevent the installation of devices that are not specifically described by any other policy setting.

If you enable this policy setting, Windows is prevented from installing, or updating the device driver for, any device that is not described by either the "Allow installation of devices that match any of these device IDs" or the "Allow installation of devices for these device classes" policy settings.

If you disable or do not configure this policy setting, Windows is allowed to install, or update the device driver for, any device that is not described by the "Prevent installation of devices that match any of these device IDs," "Prevent installation of devices for these device classes," or "Prevent installation of removable devices" policy settings.

Author's Notes:

As noted above, this option will have to be set for any of the "Allow installation of devices *" options to be effective. Otherwise the allow options are pretty much just telling Windows to allow the installation of something this is already allowed anyway.

Registry Equivalent:

Key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{55803F47-01A6-4A85-89CE-74357A125D17}Machine\Software\Policies\Microsoft\Windows\DeviceInstall\Restrictions]
```

Value if enabled:

```
"DenyUnspecified"=dword:00000001
```

Recommendations for preventing PHUKD devices from functioning

Our personal recommendation: If you want to make sure USB hardware like the PHUKD are not surreptitiously installed on a system, do the following:

1. Enable both "Allow administrators to override Device Installation Restriction policies" and "Prevent installation of removable devices".

2. Set "Display a custom message title when device installation is prevented by a policy setting" and "Display a custom message when installation is prevented by a policy setting" to something meaningful so the user knows why the hardware did not install properly so they can contact an administrator.

3. Whenever you install a new device on purpose, manually go into device manager, and install the drivers using the "Update Driver Software…" option.

Useful links:

Prevent Installation of Removable Devices
http://technet.microsoft.com/es-es/library/cc753539%28WS.10%29.aspx

List of Class IDs
http://msdn.microsoft.com/en-us/library/ff553426%28VS.85%29.aspx

Battery	{72631e54-78a4-11d0-bcf7-00aa00b7b32a}
Biometric	{53D29EF7-377C-4D14-864B-EB3A85769359}
Bluetooth	{e0cbf06c-cd8b-4647-bb8a-263b43f0f974}
CDROM	{4d36e965-e325-11ce-bfc1-08002be10318}
DiskDrive	{4d36e967-e325-11ce-bfc1-08002be10318}
Display	{4d36e968-e325-11ce-bfc1-08002be10318}
FDC	{4d36e969-e325-11ce-bfc1-08002be10318}
FloppyDisk	{4d36e980-e325-11ce-bfc1-08002be10318}
HDC	{4d36e96a-e325-11ce-bfc1-08002be10318}
HIDClass	{745a17a0-74d3-11d0-b6fe-00a0c90f57da}
Dot4	{48721b56-6795-11d2-b1a8-0080c72e74a2}
Dot4Print	{49ce6ac8-6f86-11d2-b1e5-0080c72e74a2}
61883	{7ebefbc0-3200-11d2-b4c2-00a0C9697d07}
AVC	{c06ff265-ae09-48f0-812c-16753d7cba83}
SBP2	{d48179be-ec20-11d1-b6b8-00c04fa372a7}
1394	{6bdd1fc1-810f-11d0-bec7-08002be2092f}
Image	{6bdd1fc6-810f-11d0-bec7-08002be2092f}
Infrared	{6bdd1fc5-810f-11d0-bec7-08002be2092f}
Keyboard	{4d36e96b-e325-11ce-bfc1-08002be10318}
MediumChanger	{ce5939ae-ebde-11d0-b181-0000f8753ec4}
MTD	{4d36e970-e325-11ce-bfc1-08002be10318}
Modem	{4d36e96d-e325-11ce-bfc1-08002be10318}
Monitor	{4d36e96e-e325-11ce-bfc1-08002be10318}
Mouse	{4d36e96f-e325-11ce-bfc1-08002be10318}
Multifunction	{4d36e971-e325-11ce-bfc1-08002be10318}
Media	{4d36e96c-e325-11ce-bfc1-08002be10318}
MultiportSerial	{50906cb8-ba12-11d1-bf5d-0000f805f530}
Net	{4d36e972-e325-11ce-bfc1-08002be10318}
NetClient	{4d36e973-e325-11ce-bfc1-08002be10318}
NetService	{4d36e974-e325-11ce-bfc1-08002be10318}
NetTrans	{4d36e975-e325-11ce-bfc1-08002be10318}
SecurityAccelerator	{268c95a1-edfe-11d3-95c3-0010dc4050a5}
PCMCIA	{4d36e977-e325-11ce-bfc1-08002be10318}
Ports	{4d36e978-e325-11ce-bfc1-08002be10318}
Printer	{4d36e979-e325-11ce-bfc1-08002be10318}
Processor	{50127dc3-0f36-415e-a6cc-4cb3be910b65}

SCSIAdapter	{4d36e97b-e325-11ce-bfc1-08002be10318}
Sensor	{5175d334-c371-4806-b3ba-71fd53c9258d}
SmartCardReader	{50dd5230-ba8a-11d1-bf5d-0000f805f530}
Volume	{71a27cdd-812a-11d0-bec7-08002be2092f}
System	{4d36e97d-e325-11ce-bfc1-08002be10318}
TapeDrive	{6d807884-7d21-11cf-801c-08002be10318}
USB	{36fc9e60-c465-11cf-8056-444553540000}
Windows CE USB ActiveSync Devices (WCEUSBS)	{25dbce51-6c8f-4a72-8a6d-b54c2b4fc835}
Windows Portable Devices (WPD)	{eec5ad98-8080-425f-922a-dabf3de3f69a}
SideShow	{997b5d8d-c442-4f2e-baf3-9c8e671e9e21}

3.2 Locking down Linux using UDEV

(Notes: The following examples were generated using Ubuntu Linux 10.10, kernel 2.6.35-24-generic. Other distributions may vary in exact results. Also, we know saying USB bus is redundant, but the sentences read weird without the extra “bus”)

Mitigation on Linux systems is somewhat of a different matter than on the Windows platform. While Windows has explicit policies that can be set, Linux is possibly more flexible but harder to configure. There is a way to accomplish the task, but it requires some scripting. Modern Linux kernels use Sysfs, a virtual file system meant to give information and allow control of devices that the kernel exports. There are two specific settings we are interested in: “authorized” and “authorized_default”. All USB devices should have an “authorized” option which controls whether or not the device can communicate with the system, and all USB buses should have an “authorized_default” option that controls the default state of any new USB devices that are connected to that bus. For example, if we wished to see if the device in port three of the first USB bus is active we could use the following command:

```
root@ubuntu:~# cat /sys/bus/usb/devices/1-3/authorized
1
root@ubuntu:~#
```

The 1 in the output tells us the device is authorized, 0 would mean not authorized. If we wished to see the state of “authorized_default” on the first USB bus we could use the command:

```
root@ubuntu:~# cat /sys/bus/usb/devices/usb1/authorized_default
1
root@ubuntu:~#
```

Changing these properties is as easy as piping the desired value into the virtual file (elevate to root privileges for this):

```
root@ubuntu:~# echo 0 > /sys/bus/usb/devices/usb1/authorized_default
```

The line above will disable (de-authorize) any new devices connected to USB bus one (at least until reboot). If we wished to disable (de-authorize) the device on USB bus one, port three, we use a similar command:

```
root@ubuntu:~# echo 0 > / sys/bus/usb/devices/1-3/authorized
```

Now that we know what has to be set to authorize/de-authorize USB devices, we have to script it so the authorization takes place as soon as the device is connected to the system. UDEV allows us to create rules to configure devices as they are added or removed from a system, in our case we will allow or disallow USB devices based on their identifying strings. The first thing that needs to be done is to collect identifying strings that can be used to whitelist/blacklist USB devices. The first utility that comes to mind is lsusb, which will give output resembling the following (important parts for later sections have been highlighted):

```
adrian@ubuntu:~$ lsusb
Bus 002 Device 004: ID 0a12:0001 Cambridge Silicon Radio, Ltd Bluetooth Dongle (HCI
mode)
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 004: ID 0781:5535 SanDisk Corp.
Bus 001 Device 003: ID 046d:0809 Logitech, Inc. Webcam Pro 9000
Bus 001 Device 002: ID 148f:3070 Ralink Technology, Corp. RT2870/RT3070 Wireless
Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
adrian@ubuntu:~$
```

While this gives us some information, it's not very specific for our use of writing UDEV rules. For example, the device number does not really help us much by itself, what we really need are attributes a UDEV script can key on. The lsusb's verbose option could be used (lsusb -v), and this option is useful for finding out more information about the attached devices, but the format is not as convenient for copying and pasting into UDEV scripts. For this we will use the udevadm tool. Because the Sysfs paths we will use are based on port numbers, and by default lsusb displays device numbers, we will have to do a little more work first (note: device numbers are not the same as port numbers, do not confuse the two).

A handle for any given USB device may exist in multiple places in Sysfs, but we will be using the paths inside of /sys/bus/usb/devices/. To give an example, here is a directory listing of the paths inside of /sys/bus/usb/devices/ on our test system:

```
root@ubuntu:~# ls /sys/bus/usb/devices/
1-0:1.0  1-2      1-2:1.2  1-3:1.0  2-1:1.0  2-2      usb2
1-1      1-2:1.0  1-2:1.3  2-0:1.0  2-1:1.1  2-2:1.0
1-1:1.0  1-2:1.1  1-3      2-1      2-1:1.2  usb1
root@ubuntu:~#
```

The subdirectories we are interested in are in the format (bus number)-(port):(configuration number).(interface). We will not be using the configuration and interface numbers so the shorter paths will suffice. Also, if a hub is used the port section of the directory name may be a series of port numbers separated by periods. More information about the layout of this USB Sysfs path can be found at:

http://www.linux-usb.org/FAQ.html#i6

Let's say we want more information on the wireless adapter on bus 1 and listed as device 1. We will first need to find this device's port number. Along with the previous lsusb output, lsusb's USB device hierarchy tree display option will be of use:

```
adrian@ubuntu:~$ lsusb -t
2-1:1.2: No such file or directory
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M
    |__ Port 1: Dev 4, If 0, Class='bInterfaceClass 0xe0 not yet handled', Driver=btusb, 12M
    |__ Port 1: Dev 4, If 1, Class='bInterfaceClass 0xe0 not yet handled', Driver=btusb, 12M
    |__ Port 1: Dev 4, If 2, Class=app., Driver=, 12M
    |__ Port 2: Dev 3, If 0, Class=hub, Driver=hub/7p, 12M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci_hcd/6p, 480M
    |__ Port 1: Dev 2, If 0, Class=vend., Driver=rt2870, 480M
    |__ Port 2: Dev 3, If 0, Class='bInterfaceClass 0x0e not yet handled', Driver=uvcvideo, 480M
    |__ Port 2: Dev 3, If 1, Class='bInterfaceClass 0x0e not yet handled', Driver=uvcvideo, 480M
    |__ Port 2: Dev 3, If 2, Class=audio, Driver=snd-usb-audio, 480M
    |__ Port 2: Dev 3, If 3, Class=audio, Driver=snd-usb-audio, 480M
    |__ Port 3: Dev 4, If 0, Class=stor., Driver=usb-storage, 480M
adrian@ubuntu:~$
```

We can look at this output for the device number (2) we found from the first use of the lsusb command, and then look for its corresponding bus (1) and port (1) numbers. Now that we know this information, we can use the udevadm tool to find out more information about the device, and its parent devices.

```
adrian@ubuntu:~$ udevadm info -a -p /sys/bus/usb/devices/1-1

Udevadm info starts with the device specified by the devpath and then
walks up the chain of parent devices. It prints for every device
found, all possible attributes in the udev rules key format.
A rule to match, can be composed by the attributes of the device
and the attributes from one single parent device.

  looking at device '/devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1':
    KERNEL=="1-1"
    SUBSYSTEM=="usb"
    DRIVER=="usb"
    ATTR{configuration}==""
    ATTR{bNumInterfaces}==" 1"
    ATTR{bConfigurationValue}=="1"
    ATTR{bmAttributes}=="80"
    ATTR{bMaxPower}=="450mA"
    ATTR{urbnum}=="476239"
    ATTR{idVendor}=="148f"
    ATTR{idProduct}=="3070"
    ATTR{bcdDevice}=="0101"
    ATTR{bDeviceClass}=="00"
    ATTR{bDeviceSubClass}=="00"
    ATTR{bDeviceProtocol}=="00"
    ATTR{bNumConfigurations}=="1"
    ATTR{bMaxPacketSize0}=="64"
    ATTR{speed}=="480"
    ATTR{busnum}=="1"
    ATTR{devnum}=="2"
    ATTR{devpath}=="1"
    ATTR{version}==" 2.00"
    ATTR{maxchild}=="0"
    ATTR{quirks}=="0x0"
    ATTR{avoid_reset_quirk}=="0"
    ATTR{authorized}=="1"
    ATTR{manufacturer}=="Ralink"
```

```
    ATTR{product}=="802.11 n WLAN"
    ATTR{serial}=="1.0"
<Output truncated for brevity>
adrian@ubuntu:~$
```

All of these “ATTRS” attributed can be keyed on with conditional statements in our UDEV rule scripts. If the device is still hard to find try using the “udevadm monitor” or “tail /var/log/messages” commands before and after removing and reinserting a known device to see what it reports itself as. Once we have information on the devices we wish to deal with, we can start writing UDEV rules to enable/disable them. For example, let’s create a script at:

/etc/udev/rules.d/01-usblockdown.rules

The contents of the script are as follows:

```
#Script by Adrian Crenshaw
#With info from Michael Miller, Inaky Perez-Gonzalez and VMWare

#By default, disable it.
#ACTION=="add", SUBSYSTEMS=="usb", RUN+="/bin/sh -c 'echo 0 >/sys$DEVPATH/authorized'"
ACTION=="add", SUBSYSTEMS=="usb", RUN+="/bin/sh -c 'for host in
/sys/bus/usb/devices/usb*; do echo 0 > $host/authorized_default; done'"

#Enable hub devices. There may be a better way than this.
ACTION=="add", ATTR{bDeviceClass}=="09", RUN+="/bin/sh -c 'echo 1
>/sys$DEVPATH/authorized'"

#Other things to enable
ACTION=="add", ATTR{idVendor}=="046d", ATTR{idProduct}=="0809", RUN+="/bin/sh -c 'echo
1 >/sys$DEVPATH/authorized'"
ACTION=="add", ATTR{serial}=="078606B90DD3", RUN+="/bin/sh -c 'echo 1
>/sys$DEVPATH/authorized'"
ACTION=="add", ATTR{product}=="802.11 n WLAN", RUN+="/bin/sh -c 'echo 1
>/sys$DEVPATH/authorized'"
#ACTION=="add", ATTR{idVendor}=="413c", ATTR{idProduct}=="2106", RUN+="/bin/sh -c
'echo 1 >/sys$DEVPATH/authorized'"
```

Essentially, this script de-authorizes everything by default except hubs, then authorizes individual devices based on their identifying strings, be they vendor ID, product ID, device class, product name, etc. The script above can likely be improved, and may have bugs. It is only meant to be an example script to help explain how USB policies may be set up in Linux. You can test your script with the command:

udevadm test /sys/bus/usb/devices/usb1/1-1

The “udevadm test” command will not actually apply the changes, merely show which conditional statements are tripped and what will be run when a certain device is inserted.

This section is only a very brief overview of how UDEV rules can be used to lockdown a system in regards to USB devices. For more information on UDEV rules and Sysfs please see the following links:
http://www.reactivated.net/writing_udev_rules.html
http://en.wikipedia.org/wiki/Sysfs
http://www.mjmwired.net/kernel/Documentation/usb/authorization.txt

Keep in mind that this is not a perfect defense, as it is possible to spoof many of the identifying strings a USB device reports. If an attacker knows a fair amount about what kind of devices are likely to be installed and already authorized they may be able to spoof the identifying strings. This is especially the case with PHIDS that the attacker can manipulate at a much lower level.

Thanks to Michael Miller from the PaulDotCom podcast mailing list who pointed me to some information that helped with this section (Perez-Gonzalez, 2007).

4. Tracking and scanning for malicious USB devices in Windows environments

While the author was doing research on USB hardware key loggers and programmable HIDs he learned quite a bit about USB devices that he did not know before. Besides Vendor IDs and Product IDs, some devices also have a serial number associated with them. The “Serial Number Descriptor” string is optional, but a fair number of devices such as thumb drives, removable USB hard drives, PDAs and cell phones have them. While most of these scanning and tracking details would be of most use with the “USB Mass Storage devices containing malware” and the “U3 thumb drives with "evil" autorun payloads” categories, they may still be of use against “USB Hardware Key Loggers” and "Programmable HID USB Keyboard Dongle". We will cover how scanning and tracking of USB devices can be used in the case of each category:

USB Mass Storage devices containing malware
U3 thumb drives with "evil" autorun payloads

These two categories are close enough that they can be covered together with regards to tracking and scanning. As both categories are basically traditional malware on storage, if one instance of a compromised host can be found, then its characteristics can be scanned for. As an example: If malware has been spreading through a network, and you suspect thumb drives or another type of removable USB media is the vector, you can look at a known compromised host and note the Vendor/Product IDs and serial numbers of all USB storage devices that have been attached to it. Using these identifiers a scan of the hosts on the network can be done to search for computers where a known infected drive was used, or at least for common shared devices with the known compromised system, to give an idea as to which workstations should be inspected closer. It may also lead to knowing who brought the malware into the network (Patient Zero), or who has been the biggest Typhoid Mary.

USB Hardware Key Loggers

This one is a little harder to scan for. Most USB key loggers don’t show up as a device at all to the host unless they are in “recovery mode” and that can generally be done on the attacker’s own machine outside of the reach of the victim doing the scanning. In the case of the few USB key loggers that show up as USB hubs, a search for known bad Vendor/Product IDs may be fruitful, but as discussed previously these IDs may be pretty inconspicuous leading to false positives. Perhaps a scan could be done for devices that should be reporting themselves as running at 2.0 speeds, but for some reason are throttled back to 1.0/1.1 speeds, but again we imagine there will be many false positives.

Programmable HID USB Keyboard Dongle

As the attacker can set any Vendor/Product ID they wish, scanning for these IDs may not be directly useful. Still, if the attacker has a sense of humor (I set my Vendor and Product IDs to 1313 and 0666 respectively), or uses the same IDs on all the devices they make and one of the devices is found in an organization, then those IDs could be scanned for to find other compromised hosts. Also, by constructing a database of what USB devices are attached to which host on the network, certain anomalies may be searched for. For example, an Apple keyboard plugged into a Dell desktop may stand out, or a desktop that has more than one keyboard. Still, this searching for anomalies is likely to have many false positive as under the right circumstances the user may really have two keyboards attached (this is commonly done on home media systems), or a presentation remote, barcode/magstripe scanner or game controller could represent itself as a keyboard type HID.

Finding out what USB devices are installed

There are two main spots you can pull this information from on a Windows XP/Vista/7 system:

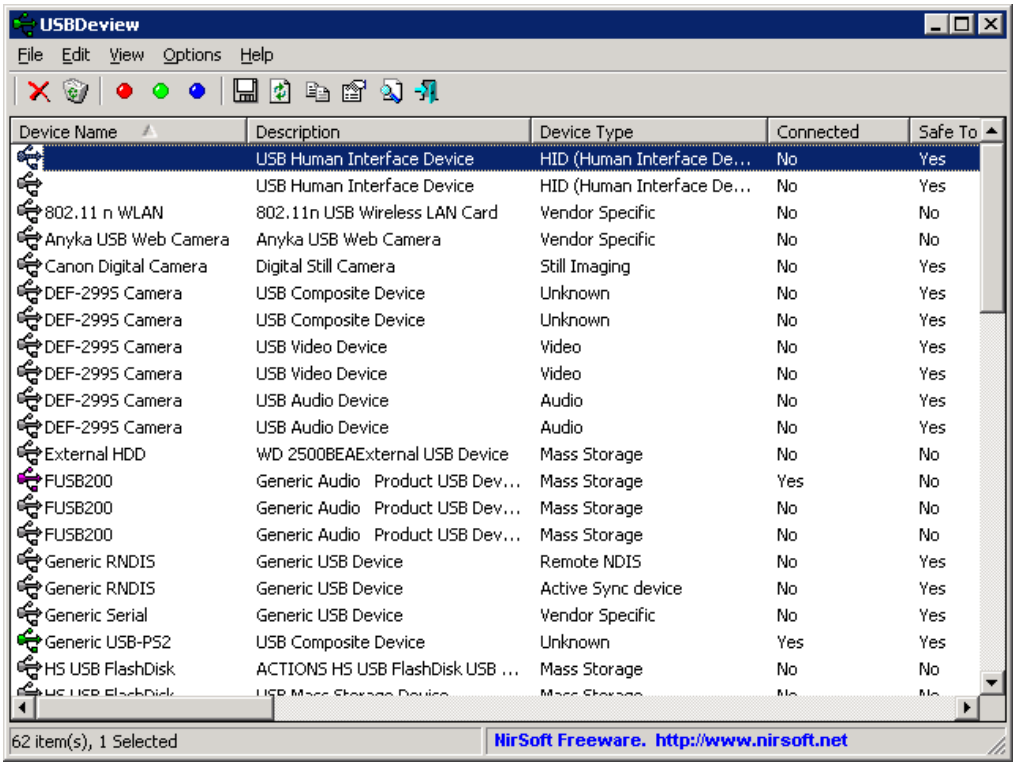
```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB
```

Contains a list of Installed USB devices, both connected and unconnected.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USBSTOR
```

Contains a list of installed USB storage devices. Looking at these registry values directly is rather cumbersome. A nicer way to look at this data for those using a Microsoft Windows platform is with NirSoft's freeware tool USBDeview which can downloaded from:

http://www.nirsoft.net/utils/usb_devices_view.html



USBDeView gives the user a ton of information about what devices are currently plugged-in, and what ones have been plugged-in previously but are not currently present.

We know where on a Windows box this USB information is stored, but is there an automated way to find it and search a network for other locations where the same USB device has been used, or store the lists of installed USB device for latter anomaly detection? Some organizations may have an asset tracking database that would be searchable for this information, but many don't have such a system in place. While in the future coding something specific for the purpose may be fruitful, current tools can be used with a little bit of scripting and data massaging to much the same effect. Most of Nir's tool have the ability to be used from across the network, and USBDeview has this option as well. All that has to be done to connect to a remote Windows box and look though its registry for USB devices is to issue this simple command:

```
USBDeview /remote \\SomeComputer
```

Now keep in mind, the person issuing the command will need to be logged in under an account with administrator privileges, and certain services have to be accessible over the network. Nir has a useful blog post on what it takes for his tools to work from across the network (Sofer, 2009).

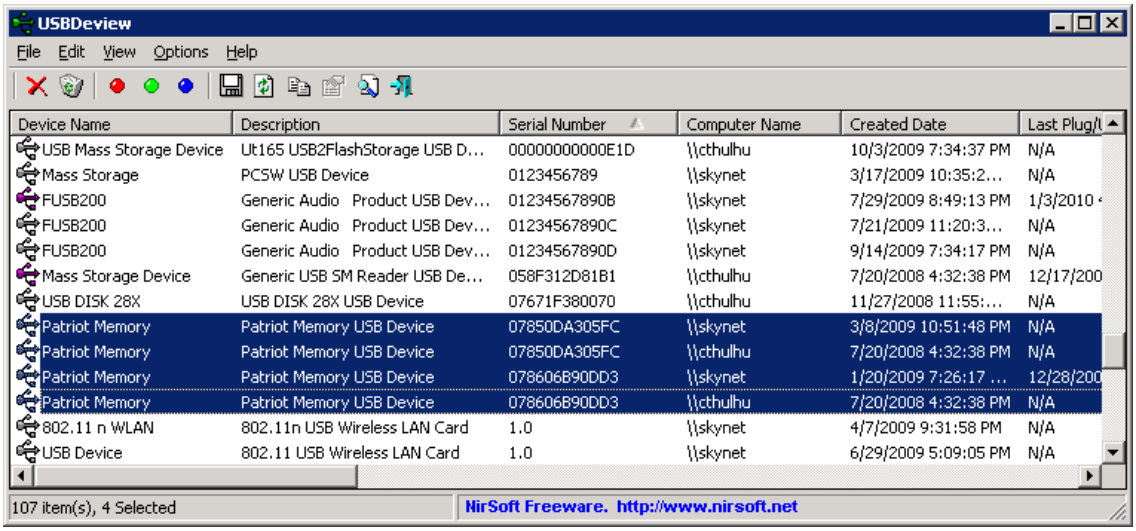
That solves the problem of dumping information from one machine, but what about a whole network? Well, it seems USBDview has an option for taking a list of machines from a text file and grabbing the USB device list from all of them. The syntax is pretty simple:

```
USBDeview /remotefile boxes.txt
```

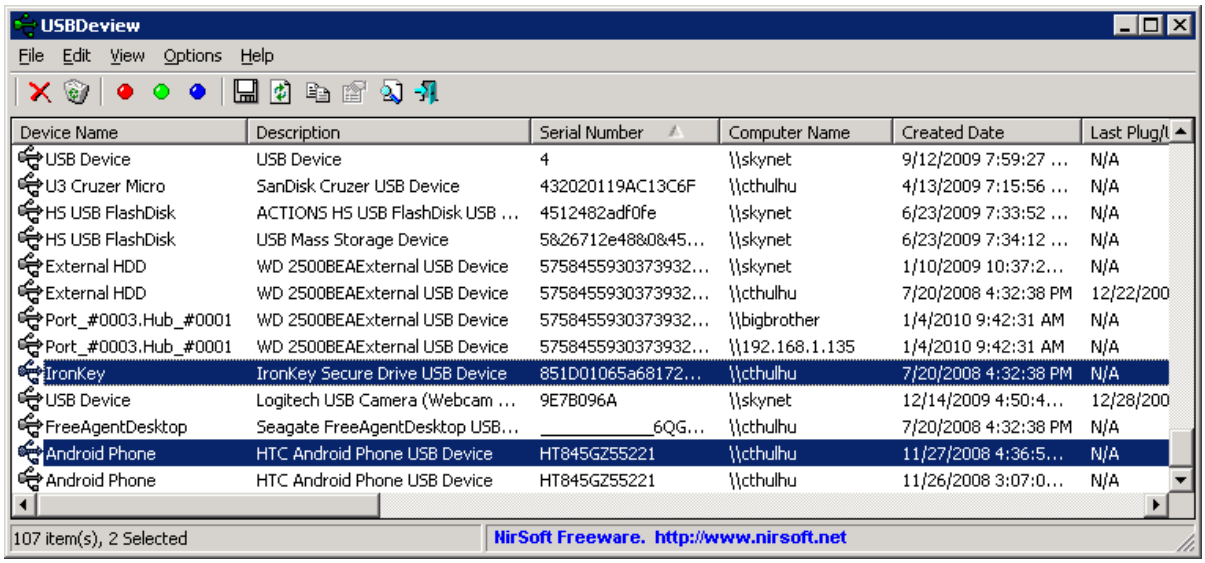
where boxes.txt is a list of computer names/IPs in UNC format. For example:

```
\\192.168.1.13
\\192.168.1.14
\\skynet
\\cthulhu
```

It takes a bit of time for the results to return, so be patient. Once the report is returned, the person doing the scan can sort by the 'Serial Number' field, look for repeating serial number by eye, and then scroll over to the 'Computer Name' field to see what computers that particular USB device has been used on. For these screen shots we've reordered the columns to put the fields we are most interested in first. The 'Created Date' and 'Last Plug/Unplug Date' fields are also useful for figuring out a time line (really handy for finding a malware Patient Zero or Typhoid Mary based on when a user was logged on last). As you can see from the first screen shot, there have been two different thumb drives from Patriot Memory that have been plugged into both the Skynet and Cthulhu workstations, but none of the other boxes scanned:



From the second screen shot you can see that an IronKey brand thumb drive and an Android based phone have only ever been plugged into the workstation named Cthulhu:



This sort of information can be erased, but local attackers rarely think of the tracks that their USB devices leave behind, and certain privilege levels are required to erase these logs.

Now the reader may be thinking: sorting and looking though the table for matching USB serial numbers by hand may be fine for when there are only four workstations, but what if the network to be searched has a lot more workstations/servers? Luckily, Nir implemented the ability to save his tool's output to many different file formats, including CSV (Comma Separated Values). Pretty much any database tool (MS Access for example) can import a CSV file, and from there it's just a few SQL queries away from finding the devices and information the person doing the scan is seeking. For example, if the scanner wanted to look though the output for just the serial number 07850DA305FC an SQL query similar to this one may help:

```
SELECT Myoutput.*
FROM Myoutput WHERE Myoutput.[serial] = '07850DA305FC' ;
```

To dump the findings to a CSV the person doing the scan could take the results from the GUI, select all of the records (Ctrl-A), then choose the save icon and pick CSV from the drop down, but there is an easier way if they only intend to import the results into a database for sorting and searching. USBDeview can be instructed to dump the output to a file without ever bringing up the GUI with this simple command:

```
USBDeview /remotefile boxes.txt /scomma myoutput.csv
```

With a little automation, regular reports can be made, and all sorts of anomalies can be queried for. Research is still ongoing into finding better ways to track and sort this information, so if the reader knows of any good free or open source asset management systems that log USB serial numbers/identifier strings, or is interested in coding something to help automate these types of searches, please contact the author.

5. Possible extensions to the USB standards

While it is possible for modern operating systems to block USB devices based on identifiers such as Vendor or Product ID, these solutions are less than optimal. Vendor and Product IDs are easy to spoof for those who can develop their own USB devices. Granted, the attacker will have to have a greater level of knowledge concerning what resources may be in use at a facility, but there are ways this can be found out, especially by insiders or someone in the supply chain. USB devices that support serial numbers are somewhat better when it comes to uniqueness, but these serial numbers may still be predictable.

Possible future solutions to the problem could include pairing of devices in a way similar to the current Bluetooth standard, or allowing the user/operating system to write a unique ID to the device that can be used to identified it in the future. These solutions still may not be fool proof. If someone has the physical access to install the USB device in the first place they may be able to extract the approved unique identifiers or pairing keys from the operating system in much the same way that WPA/WEP keys can be extracted from a system. Also, it is doubtful such security options will be used except for in the most locked down environments as they greatly decrease the user-friendliness of USB devices. In environments that would merit such tight security controls, it is hoped that they would have appropriate physical security and user educations to keep malicious USB devices from being connected in the first place.

6. Conclusions

The convenience of USB from a user's perspective also makes it convenient for an attacker. Anti-malware tools can help against some attack vectors ("U3 Thumb drives with "evil" autorun payloads" and "USB mass storage containing malware") but are not complete solutions because:

1. Custom created malware may not be detected by Anti-malware systems.
2. There are attack vectors using USB devices beyond just simple malware on removable storage.

Both storage and non-storage based USB attack vectors can be mitigated by using the policy options built into Windows (GPO/Local Security Policy) and Linux (UDEV and USB Authorizations) but these may fall short because:

1. Identifying strings that the policies key on to decide if a device should be allowed or not can be spoofed.
2. Most users are unlikely to want to bother with authorizing each USB device they connect, though locked down government and corporate environments may wish to take such precautions.

There is hope that in the future the USB spec could be extended to allow for less spoofable identifying strings, but addition of this feature is largely out of the end user's hands, and if it becomes available is not likely to be used anyplace but the most locked down environments that should be using already existing mitigations.

Luckily, for the average computer users, current USB attack vectors are either too expensive or take too much customization to be widely deployed against them, at least not in the same

way as phishing and drive by installs. If the average user is going to be the victim of a malicious USB device, it is likely to be either personally motivated or a prank (beware of geeks bearing gifts). There just does not seem to be a good profit motive at this time to attack the average home user.

Government and corporate systems are a different matter however. In these environments there may be enough valuable information for an attacker to justify the investment of time and money it would take to pull off an attack. The cost of $20 in hardware and a few hours of time in programming and research is prohibitively high to attack most home users, but is a pittance compared to the profits that could be gained from information gleaned by compromising government or corporate systems. Government and corporate systems may be well advised to do the following:

1. Ensure adequate physical security. If a USB device cannot be plugged into a sensitive system, it's not much of a threat.
2. Train users with access to sensitive systems to be wary of plugging in unfamiliar USB devices. This includes plugging them into their home computer if that computer is used to access sensitive resources. Physical security that keeps outsiders away is not of much use if an insider can be easily convinced to plug in a USB device for the attacker.
3. To compensate for those times when physical security and user training fail, implement policy's on sensitive systems that at least make it harder for rogue USB devices to be installed by mistake. If a secretary has to ask an admin to install it before they can get a USB device to work, at least that is two people whose security training has to fail instead of just one.

Hopefully these measures will go a long way in mitigating the threats posed by malicious USB devices.

Special thanks to:

Dr. Minaxi Gupta for being my advisor on the project for my Master in Security Informatics studies at Indiana University

The Tenacity Institute, especially Dr. David Comings and Gene Bransfield, for helping to finance the project and speaking engagements

Bill Swearingen, David Porcello, Alberto Partida, Mike Patterson, Ben Smith, Dennis Sutch, Jeff Barone and Bart Hopper for proofreading and giving technical comments

Works Cited

Bustamante, P. (2010, March 8th). *Vodafone distributes Mariposa botnet*. Retrieved 8 10, 2010, from Panda Research Blog: http://research.pandasecurity.com/vodafone-distributes-mariposa/

Butturini, R. (n.d.). *Using the Hak5 U3 Switchblade as an Incident*. Retrieved 1 23, 2011, from http://www.irongeek.com/i.php?page=videos/pn12/russell-butturini-using-the-hak5-u3-switchblade-as-an-incident-response-and-forensics-tool

Crenshaw, A. (2007). *Irongeek's Key logger Research*. Retrieved 11 8, 2010

Crenshaw, A. (2007, 10 17). *Thumbscrew: Software USB Write Blocker*. Retrieved 11 8, 2010, from Irongeek.com: http://www.irongeek.com/i.php?page=security/thumbscrew-software-usb-write-blocker

Crenshaw, A. (2010, 8 3). *Programmable HID USB Keystroke Dongle: Using the Teensy as a pen testing device*. Retrieved 8 17, 2010, from Irongeek.com: http://www.irongeek.com/i.php?page=security/programmable-hid-usb-keystroke-dongle

Darkreading. (2006, 6 7). *Social Engineering, the USB Way* . Retrieved 8 11, 2010, from Darkreading: http://www.darkreading.com/security/perimeter/showArticle.jhtml?articleID=208803634

HAK5 Wiki. (n.d.). *USB Switchblade*. Retrieved 8 11, 2010, from HAK5: http://www.hak5.org/w/index.php/USB_Switchblade

Johansson, J. M. (2008, 1). *Island Hopping: The Infectious Allure of Vendor Swag*. Retrieved 8 11, 2010, from Microsoft Security Watch: http://technet.microsoft.com/en-us/magazine/2008.01.securitywatch.aspx

KeeLog. (n.d.). *KeyDemon Wi-Fi* . Retrieved 8 11, 2010, from KeeLog: http://www.keelog.com/wifi_hardware_keylogger.html

Leyden, J. (2010, 7 16). *Windows Shortcut Flaw underpins power plant Trojan*. Retrieved 8 10, 2010, from The Register : http://www.theregister.co.uk/2010/07/16/windows_shortcut_trojan/

Microsoft . (2006, 1 5). *Microsoft Security Bulletin MS06-001*. Retrieved 8 10, 2010, from Microsoft TechNet: http://www.microsoft.com/technet/security/Bulletin/ms06-001.mspx

Microsoft. (2009, 9 11). *How to disable the Autorun functionality in Windows*. Retrieved 11 8, 2010, from Microsoft Support: http://support.microsoft.com/kb/967715

Microsoft. (2010, 7 1). *How to disable the Autorun functionality in Windows*. Retrieved 8 11, 2010, from Microsoft Support: http://support.microsoft.com/kb/967715

Microsoft. (2010, 7 1). *Update to the AutoPlay functionality in Windows*. Retrieved 8 11, 2010, from Microsoft Support: http://support.microsoft.com/kb/971029

Perez-Gonzalez, I. (2007). *USB Authorization*. Retrieved 11 8, 2010, from MJM Wired: http://www.mjmwired.net/kernel/Documentation/usb/authorization.txt

Schneier, B. (2005, 11 17). *Sony's DRM Rootkit: The Real Story*. Retrieved 8 11, 2010, from Schneier on Security: http://www.schneier.com/blog/archives/2005/11/sonys_drm_rootk.html

Sofer, N. (2009, 10 22). *How to connect a remote Windows 7/Vista/XP computer with NirSoft utilities*. Retrieved 11 8, 2010, from NirBlog: http://blog.nirsoft.net/2009/10/22/how-to-connect-a-remote-windows-7vistaxp-computer-with-nirsoft-utilities/

SophosLabs. (2006, 10 17). *SophosLabs*. Retrieved 8 10, 2010, from Malware shipped on Apple Video iPods : http://www.sophos.com/pressoffice/news/articles/2006/10/ipod-ships-with-virus.html

Wirelesskeylogger. (n.d.). *Wirelesskeylogger*. Retrieved 8 11, 2010, from Wirelesskeylogger: http://www.wirelesskeylogger.com/

Zetter, K. (2008, 1 31). *Digital Photo Frames and Other Gadgets Infected with Malware*. Retrieved 8 10, 2010, from Wired: http://www.wired.com/threatlevel/2008/01/digital-photo-f/