



# USBWall: A novel security mechanism to protect against maliciously reprogrammed USB devices

Myung Kang and Hossein Saiedian

Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA

## ABSTRACT

Universal Serial Bus (USB) is a popular choice of interfacing computer systems with peripherals. With the increasing support of modern operating systems, it is now truly plug-and-play for most USB devices. However, this great convenience comes with a risk that can allow a device to perform arbitrary actions at any time while it is connected. Researchers have confirmed that a simple USB device such as a mass storage device can be disguised to have an additional functionality such as a keyboard. An unauthorized keyboard attachment can compromise the security of the host by allowing arbitrary keystrokes to enter the host. This undetectable threat differs from traditional virus that spreads via USB devices due to the location where it is stored and the way it behaves. We propose a novel way to protect the host via a software/hardware solution we named a USBWall. USBWall uses BeagleBone Black (BBB), a low-cost open-source computer, to act as a middleware to enumerate the devices on behalf of the host. We developed a program to assist the user to identify the risk of a device. We present a simulated USB device with malicious firmware to the USBWall. Based on the results, we confirm that using the USBWall to enumerate USB devices on behalf of the host eliminates risks to the hosts.

## KEYWORDS

SB security; physical security; transient device; Beaglebone black; human input devices (HID) attack

## 1. Trust in USB standards

As the Universal Serial Bus (USB) interface gained popularity thanks to its plug-and-play capability and small form factor, operating systems have started to support more types of devices. Most USB devices have hot-swapping and plug-and-play capability, which allows for rapid device initialization as soon as it is plugged in. Combined with the size of the NAND flash and native support that allows it to be activated without additional software, the flash drive became the most popular USB device (Appavoo et al., 2003; Intel Corporation and Microsoft Corporation, 1999).

To provide plug-and-play, the underlying protocol is designed to minimize user interaction in dynamically allocating system resources (Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, 2000). Meanwhile, during this time, the host must trust the device's initial information to which it initializes. However, leveraging on the large driver base of modern operating systems and along with users' inaccurate models of threat

possibilities that make users believe they are safer than they actually are (Tetmeyer & Saiedian, 2010), a new threat appeared. The threat, coined BadUSB by Security Research Lab, exploits the trusting nature of the current USB specification to allow a device to present itself as a different, potentially malicious, type of device (Harman, 2014; Security Research Labs, 2014).

For example, a flash drive may establish the USB handshake as a keyboard. Exploiting users' inaccurately perceived sense of safety, a seemingly small USB drive may appear to be a safer device than a physical keyboard. Being granted keystroke access to a system effectively adds an attack surface by allowing the attacker to run arbitrary commands with the privilege of the currently logged-in user (Security Research Labs, 2014). This is a particularly alarming finding because of the widespread support of USB in everyday devices. Not only computers, but also all USB-enabled devices that support USB connectivity, are susceptible to this attack. The added

attack surface is significant because it is platform-independent. Because the attack takes advantage of the USB standards itself, it can potentially affect all USB-capable devices such as automobile, home appliances, and any other devices with USB ports. When exploited, it can cause unintended behaviors ranging from annoying to compromised security measures.

This discovery differs from traditional security vulnerabilities such as Stuxnet, which used USB drives as one of its primary means to propagate (Falliere, Murchu, & Chien, 2011). In the case of Stuxnet, the virus hides within the storage area of the device. BadUSB, on the other hand, is affected and executed within the firmware area of the device. Therefore, no commercially available anti-virus can detect the existence of malicious firmware. Because the firmware exploits the low-level USB transaction, this new vulnerability can affect all USB-capable devices. Primary methods of protection against virus on removable devices have been to scan the files before read and write operations. Therefore, there is currently no available protection for USB devices with malicious firmware.

Potential vulnerabilities of the USB specifications have been well discussed, such as lack of USB attestation (Zhaohui, Johnson, & Stavrou, 2012) and leaking confidential confirmation using unintended channels (Clark, 2009; Clark, Leblanc, & Knight, 2011). Because of the risks, many organizations resort to well-written legal policies and try to shape employees' behavior. Although such policies prohibit the usage of non-trusted devices, and they do not totally stop non-trusted devices, they usually give the organization legal rights after an incident happens (Tetmeyer & Saiedian, 2010).

This article focuses on developing countermeasures against BadUSB devices, and evaluating their effectiveness. As the attack takes place at the lower-level operations of the USB controller, no measure exists for the host operating system to distinguish the infected re-enumerating from physical plug-in events. Therefore, we developed USBWall as a novel way to add a protection layer with an additional middleware between the host and devices.

### **1.1. Significance of sandboxed USB transactions**

In the world of the cybersecurity arms race that is growing rapidly (Rueter, 2011), our research explores a practical method to counter the newly found threat of USB devices with malicious firmware. Because of the nature of the vulnerability, which operates at the low level of USB standard, the threat can potentially affect all USB-capable devices. We believe that the current standard implies too much trust when initializing the device, and the trust must be displaced to achieve a secure USB environment.

Although there are not many existing studies regarding this new risk of USB devices with malicious firmware, other risks of user devices including USB storage devices (transient storage devices, TSDs) are well discussed. In this section, we discuss the existing studies about user devices' risk as well as proposed standards that attempt to make the USB interfacing safer by authorizing and authenticating USB devices. Several proposals have been submitted and approved. However, they were not implemented in mainstream operating systems to offer effective protection for users.

As more manufacturing is outsourced to other countries (Mitra, 2013), the risk exists for an unauthorized change to be made to the product design at the manufacturing phase. Even if the product meets all specification requirements, it has a possibility to operate unexpectedly under certain predefined circumstances. For instance, encryption modules which are mass-produced in distant factories can carry risks such as allowing unauthorized parties to access the private key on the chip's memory (Jin, 2012).

The term "hardware trojan" refers to an unauthorized change in hardware components to allow bypassing or weakening of designed behavior manufactured offshore (Karabarounis & Neiman, 2013). A device with one or more such components could cause unexpected behavior. Increasing outsourcing of device fabrication foundries contributes to the heightened possibilities of malicious circuits being inserted (Oshri, Kotlarsky, & Willcocks, 2015; Jin, 2012). Such attacks would insert unauthorized circuits to be activated at a later time. Such modifications can cause unexpected behaviors such as a disclosure of secret

keys in the encryption module, returning a false result, or complete destruction of the module itself.

In addition to hot-plug capability and smaller form factors, USB flash drives have quickly become the popular choice for removable storage device. While they provide users portability and convenience, incorrect user-perceived notions of security have allowed a worm such as Stuxnet to propagate and even reach stations not connected to the network (Tetmeyer & Saiedian, 2010; Falliere et al., 2011). Reacting to viruses such as Stuxnet and its variants, endpoint security programs scan files before reading and writing. This had been proven effective in providing protection to operating systems from malicious codes that attempt to execute without the user's consent.

The users are the most important entity because they are the ones who have the physical access to the protected systems. The inaccurate idea of a device by a user, misjudged by its physical appearance, often leads to security threats (Tetmeyer & Saiedian, 2010). For example, an iPod may be seen as a music player that needs charging using a corporate computer. However, when it connects, it attempts to establish connections to the computer, which could lead to violation of company policy. A small honest mistake like this could turn into an organization-wide security threat by adding just enough attack surface. The innocuousness of a TSD and its portability, along with the inaccurately perceived sense of security by the user, creates a disparity between the actual and perceived security. This is important context for our research. Our research assumes that the user is aware and suspicious of unknown USB devices.

In addition to the inaccurate model of security and the general notion of a hardware trojan, unintended channels serve to extract data using unusual devices such as keyboards and USB speakers. Unintended channels utilize data paths that are designed to carry control data for the devices

(Clark et al., 2011; Clark, 2009). Such features would be inserted in the manufacturing phase. Then, they would interact with the software counterpart on the target system to extract data. In Clark's research, an unsuspecting device such as a keyboard with modified firmware can collect and store sensitive data by tracking the LED state of Num Lock, Caps Lock, and Scroll Lock turn on and off. Although the LED activity is visible, this data path is generally not monitored by security software products. In the same research, Clark also explores the possibility to exfiltrate data via a device such as a USB speaker. By utilizing a non-audible data channel such as WAVEFORMATTEXTENSIBLE, Clark succeeds in showing that it is very possible to steal data via a set of speakers.

It is not only the owner's device that poses risk. An innocuous request from others to plug in their USB device to charge it could be just as risky as plugging in a flash drive with Stuxnet. Similar to the incorrect model of security in the previous section, users often consent to plugging in unchecked USB devices based on the relationship with the requestor. For example, a proof-of-concept named Pod Slurping was released to the public in 2005 (Usher, 2005). The concept of Pod Slurping effectively proves that an innocuous device such as a music player can have a malicious intent programmed so that the attacker can access the data on a protected system. Leveraging on the inaccurate model of security (Tetmeyer & Saiedian, 2010), consented use of user devices poses great threat to computing safety.

We propose a novel way to sandbox USB enumeration, which is the root cause of the vulnerability on which the malicious firmware attack depends. The attack exploits the nature of USB devices as shown in Figure 1. Only the mass storage area is visible to the user and the operating system. The attack executes at the firmware level,

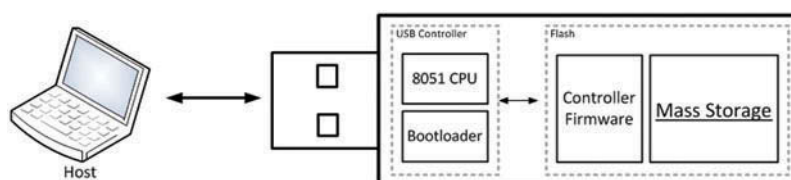


Figure 1. Diagram of USB flash drive components.

making it impossible to detect before the malicious code is executed. BeagleBone Black (BBB) embedded computer is used as a hardware platform, and an open-source project USBProxy is used as a software platform to enable a complete USB sandbox environment.

## 1.2. Research methodology

The threats arising from maliciously reprogrammed USB devices are possible because of unnecessary and excessive trust placed on devices by hosts, as well as the automatic installation of common device drivers. We believe that reducing the implied trust is the key to protecting the host. To test effective ways to break the trust between the host operating system and USB devices, the current USB specification is discussed in detail to precisely locate the best layer to displace the trust. To decrease the level of trust, we develop a set of programs called USBWall, which automates several operating systems' native commands and open-source projects such as USBProxy. These programs are designed to eliminate the implied trust between the host operating system and the device. Figure 2 shows how USBProxy (Spill & Stasiak, 2014) operates on BBB, which is one of the key components of the USBWall. A crucial part of the USBWall that handles the actual data is USBProxy by Dominic Spill. USBProxy relays USB data traffic from a device to a host using gadgetFS. Because USBProxy is launched with parameters of the device's specific VID and PID, if the device attempts to re-enumerate, USBProxy needs to relaunch. The flow is terminated automatically.

To test a BadUSB-like device, the current project developed a sample BadUSB device by using Harman's program (Harman, 2014). The device will have a different firmware from the manufacturer's, then be presented to a host in several configurations. Test results will be gathered by comparing the product, USBWall, with other commercially available antivirus products.

## 2. Related work in authenticated USB uses

Attack scenarios originating from user devices are well known. From floppy disks to iPods to USB devices, the full potentials of computer peripherals are also well recognized by security professionals, while most ordinary computer users are unaware (Arce, 2005). Historically, user devices have always carried risks of executing unexpected behavior on the host device. As modern technology develops at a stunning rate to reduce the physical size of user devices, the capacity and capabilities of the devices become more sophisticated. What were merely media on which to hold bits are now small reprogrammable computers with interfacing capability to their embedded storage flash chips.

### 2.1. The inherent trust of the USB standard

The USB specification is written with the intention of minimizing user intervention during device initialization (Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, 2000). Consequently, the complexity of hardware required to be USB-compliant devices became more sophisticated than older interfaces such as DB-9 and DB-25. Contrary to the handshake transactions of older interfaces, which are

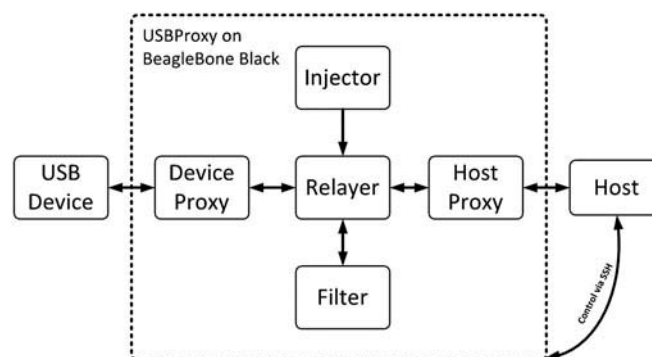


Figure 2. USBProxy architecture.

handled at the application level, USB specifications mandate that the host controller and slave controller establish the initial contact using a predefined series of electronic signals handled at the hardware level. More details about USB enumeration are discussed in Section 3. In addition to allowing for easier device initialization, to support a wider range of peripherals, the slave controller must be versatile enough to accommodate different types of transactions. This means that there must be a controller on the host side to determine the type of connected devices. The host controller initializes the device to the type of device based solely on the information sent by the device.

Unfortunately, the current USB specification implies trust between host and device by not specifying a way to attest or authenticate the device. We argue that properly placed trust is imperative to achieve safer computing. Several existing standards can affect the plausibility of the malicious firmware attack. We categorize them as nontechnical, software, and hardware measures.

## 2.2. Nontechnical measures

### 2.2.1. Policy enforcement

A very limited number of countermeasures that mitigate this newly found threat exist. However, there are nontechnical controls such as the payment card industry data security standard (PCI DSS) that could help reduce the exposure from devices with malicious firmware by mandating certain physical controls. PCI DSS disallows almost all external computer interfaces

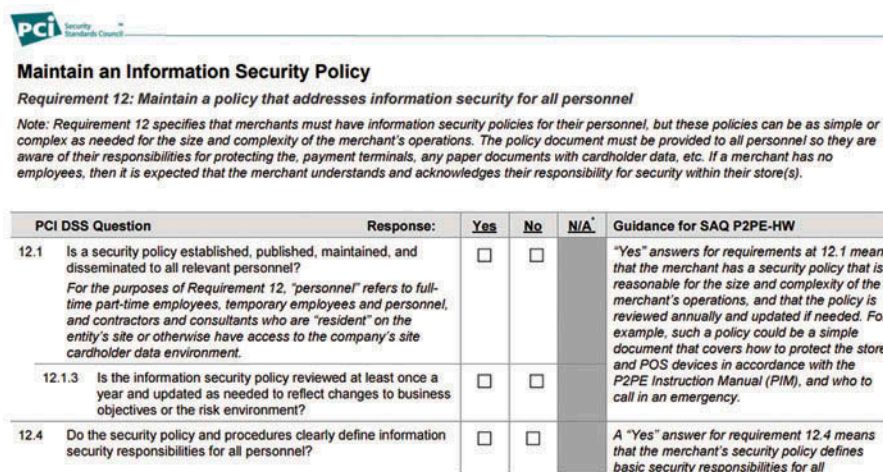
except when it is absolutely necessary and no other alternative interfacing is available. In such a case, compensating control must be declared and approved (PCI Security Standards Council, 2015). Figure 3 shows the example of PCI DSS's self-assessment questionnaire mandating policy review.

Furthermore, employee handbooks and acceptable-use policies are also used to limit physical access to the protected system to only authorized individuals. Such policies attempt to minimize the likelihood of unknowingly plugging in an unauthorized USB device. However, those policies fail to provide technical measures that prevent such attack from executing, although they provide sound legal ground to assist with legal proceedings.

### 2.2.2. Public awareness

The possibility of exploitation from USB devices with malicious firmware was announced at Black Hat USA 2014. A Security Research Labs (SRL) presentation urged the USB controller manufacturers to mitigate the problem. They demonstrated the possible use cases of the vulnerability if a USB device is infected with malicious firmware. A device with malicious firmware, named a BadUSB, was shown to appear as a completely different type of device than its physical characteristics indicate. At the time, SRL did not disclose the details of how their demonstration BadUSBs were created. Notwithstanding such a gesture, the problem remained unattended.

Later that year, Harman made the code to reprogram the firmwares of certain types of USB



**PCI Security Standards Council**

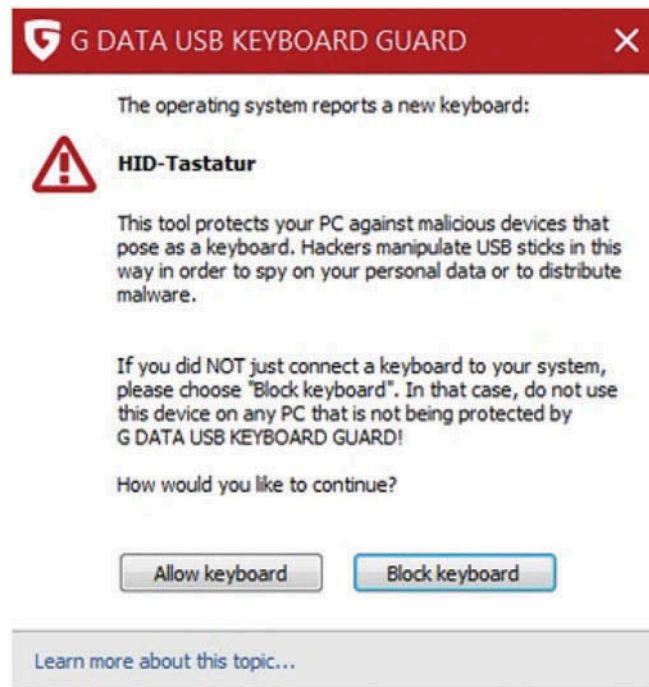
**Maintain an Information Security Policy**

**Requirement 12: Maintain a policy that addresses information security for all personnel**

*Note: Requirement 12 specifies that merchants must have information security policies for their personnel, but these policies can be as simple or complex as needed for the size and complexity of the merchant's operations. The policy document must be provided to all personnel so they are aware of their responsibilities for protecting the, payment terminals, any paper documents with cardholder data, etc. If a merchant has no employees, then it is expected that the merchant understands and acknowledges their responsibility for security within their store(s).*

PCI DSS Question	Response:	Yes	No	N/A	Guidance for SAQ P2PE-HW
12.1 Is a security policy established, published, maintained, and disseminated to all relevant personnel? <i>For the purposes of Requirement 12, "personnel" refers to full-time part-time employees, temporary employees and personnel, and contractors and consultants who are "resident" on the entity's site or otherwise have access to the company's site cardholder data environment.</i>		<input type="checkbox"/>	<input type="checkbox"/>		"Yes" answers for requirements at 12.1 mean that the merchant has a security policy that is reasonable for the size and complexity of the merchant's operations, and that the policy is reviewed annually and updated if needed. For example, such a policy could be a simple document that covers how to protect the store and POS devices in accordance with the P2PE Instruction Manual (PIM), and who to call in an emergency.
12.1.3 Is the information security policy reviewed at least once a year and updated as needed to reflect changes to business objectives or the risk environment?		<input type="checkbox"/>	<input type="checkbox"/>		
12.4 Do the security policy and procedures clearly define information security responsibilities for all personnel?		<input type="checkbox"/>	<input type="checkbox"/>		A "Yes" answer for requirement 12.4 means that the merchant's security policy defines basic security responsibilities for all

Figure 3. PCI DSS self-assessment questionnaires v2.0 sample.



**Figure 4.** G Data USB Keyboard Guard notification.

flash drives available to the public at SchmooCon 2014 (Harman, 2014). He confidently announced that his action hoped to bring manufacturers to fix the vulnerability quickly. He believed that public awareness is the key to resolving the issue. Although this approach did not provide a technical means to protect against a BadUSB, Harman's presentation made many users aware of the possibilities. These announcements were covered by several news outlets, alerting a more general audience to be aware of the risk (Brandom, 2014; Mamiit, 2014; Spector, 2014).

### 2.3. Proposed standard: IEEE 1667

To mitigate the risks from TSDs as well as unintended channels, a number of standards and modifications of USB have been proposed to attempt to authenticate and authorize a device (IEEE 1667 Working Group, 2010; Rich, 2007; Verma & Singh, 2012; Zhaohui et al., 2012). Authentication in a USB means the host and device can validate each other, and authorization means the host accepts only a predefined functionality from devices. Current practice of using vendor ID (VID) and product ID (PID) provide only limited means of protection, as they are easy to spoof and

provide only the device and manufacturer information. IEEE 1667 describes a complete way to provide both features. For example, using a concept of silos, an IEEE 1667-compliant host accepts only preauthenticated devices. As such, IEEE 1667 was proposed and approved in 2007 (IEEE 1667 Working Group, 2010). However, it is hardly used in modern operating systems and devices. Although it was proposed and announced to be implemented to Windows 7 in 2008, there do not seem to be any IEEE 1667-compliant devices in the general market.

### 2.4. Software measures

Shortly after the discovery of the risk of devices with malicious firmware at Black Hat USA 2014 (Security Research Labs, 2014), G Data published a program that traps USB keyboard insert events (G Data, n.d.). The program monitors all USB insert events, then the program is activated if the new device inserted is a HID keyboard. A pop-up notification as shown in Figure 4 appears for the user to decide.

Depending on the choice, the insert event is either allowed or blocked by the console user. If allowed, the new hardware is uninterrupted and

results in a successful device initialization. If not allowed, the new hardware is not initialized and is blocked from future insert. While the Keyboard Guard offers excellent protection without any modification to Windows operating systems, its protection is limited to HID keyboards only. Considering that reprogrammed USBs can pose as any device to any operating system, it does not offer full protection for users against malicious devices posing as something other than a keyboard.

The implementation of the IEEE 1667 standard manifested in certain Windows products under the name of Enhanced Storage Access (Microsoft, *n.d.*). However, the standard is rarely known to general users. Other than the lack of IEEE 1667-compliant devices in the market, we are unable to conclude why it was not fully accepted and implemented by operating system manufacturers.

### **2.5. Time to untrust USB**

We believe that distrusting all devices is the only way to protect the host fully from malicious USB devices. Although similar standards have been proposed, they were hardly implemented, thanks to the USB specification that requires a device to advertise its capabilities in order to initialize, we believe that monitoring the result of enumeration process will provide users a chance to block it from launching its attack.

There still exists a need for a technical solution that protects from all types of devices with malicious firmware. Focusing on the layer and stage of the USB handshake at which the BadUSB exploit occurs, we believe that the only effective way to inspect a device for the presence of malicious firmware is to prevent the host USB controller from handling any enumeration until it is deemed safe.

We propose a novel way to protect the host operating system by disallowing the enumeration yet allowing the enumeration details to be gathered. USBWall uses BBB (Coley, 2013), a low-cost open-source computer, to act as middleware to enumerate the devices on behalf of the host. Like G Data Keyboard Guard (G Data, *n.d.*), a newly inserted device is confirmed by the user before it is initialized. Unlike G Data Keyboard Guard, USBWall will read all types of

USB devices. All USB devices remain uninitialized until the user verifies and confirms a specific device. At the small expense of delayed initialization, we achieve great security and protection against BadUSB devices. By not trusting all newly plugged in devices, a firmware must present its intentions before the user accepts it. While similar protocols have been proposed (IEEE 1667 Working Group, 2010; Zhaohui et al., 2012), they require significant changes within the operating system and the devices. The solution discussed in this article requires no changes to the operating system or the device.

## **3. USBWall: An effective middleware protection**

USB devices infected with malicious firmware can be detected only by their discrepant intention at the enumeration stage. Due to the inherent risk that user devices carry, several nontechnical measures already exist to hinder the BadUSB-type attack. While some attempt to mitigate the risk from user devices by disallowing the external devices entirely, some try to educate the users about the risks. With a precise understanding of the key stage of which BadUSB takes advantage, offering the user a chance to verify a suspicious device can effectively prevent the BadUSB attack. As the device initialization never takes place on the protected host computer, the malicious firmware is not executed.

### **3.1. Identification and characteristics of BadUSB devices**

As more research on USB devices' potential capability progressed, researchers found the new threat of an innocuous USB device. Unlike Stuxnet, which resides in the storage area of a device, the new threat lives in the firmware area. For example, by reprogramming the firmware, a USB flash drive can act as a different type of device such as keyboard or network adapter. As only firmware is modified, BadUSB is not at all distinguishable by its physical appearance. In a world with higher risk of hardware trojans as more manufacturing is

outsourced (Jin, 2012), this poses a great threat to users and renders current antivirus approaches entirely useless because the infected firmware storage is inaccessible. The transactions that occur, albeit reprogrammed, remain legitimate in USB specifications. Therefore, the operating system is unable to tell the difference between a physical plug-in and re-enumerating, and will attempt to accommodate the device with a matching driver. The threat discussed in this article was first publicized by Security Research Labs (Security Research Labs, 2014). At Black Hat USA 2014, Nohl demonstrated that masquerading as a different device was possible (Security Research Labs, 2014). Nohl suggests firmware signing to prevent unauthorized change, but this approach will render the device unusable upon unauthorized change of the code. In this article, we discuss a method to determine whether a device is BadUSB. Using the method, we hope to further develop a way to protect a host operating system before it is exposed to the device.

### 3.1.1. USB enumeration and plug-n-play

When a USB device is plugged in, a series of transactions happen between the host controller and the device [Future Technology Devices International Limited (FTDI), 2009]. Figure 5 shows the order of events during the enumeration.

For our research, we focus on what happens after the host controller assigns the address.

Once the device has an address, it advertises itself to the host controller of its capabilities. Assuming a proper driver exists, the device is ready to be initialized. In such a case, the device is initialized at that point with no more user interaction. The fact that a device can have more than one feature (interface), and can be initialized with none to very limited amount of user intervention, enables BadUSB to be effective on most systems today. For example, an innocuous USB storage device can turn into a keyboard when predefined criteria are met. Thus, we seek for a way to obtain the device information to allow users to view, and decide whether or not to initialize.

### 3.1.2. Identification of BadUSB devices

To assess reliably the risk of a suspected BadUSB device, an actual enumeration process must occur. However, most operating systems, if a matching driver is available, load the driver automatically. This is precisely what BadUSB attempts to achieve. For example, by emulating a keyboard, a suspected device can start sending keystrokes as soon as the operating system finishes loading the keyboard driver.

Identification of a BadUSB device may be possible if an unexpected device behavior is observed after inserting. Any behaviors inconsistent with

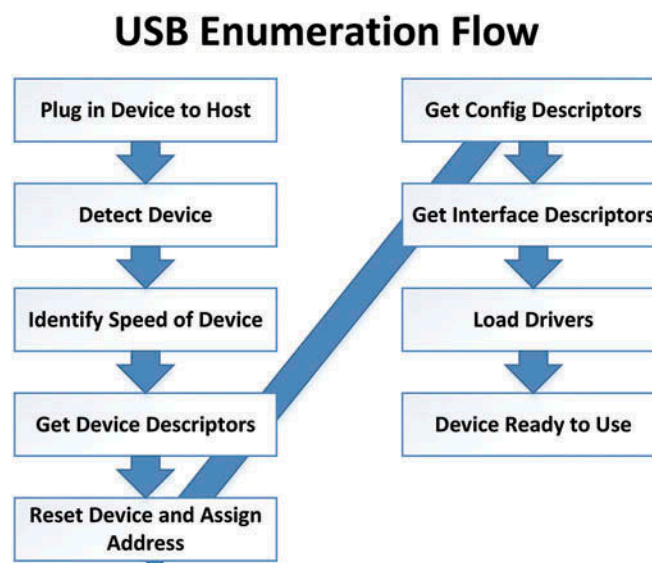


Figure 5. USB enumeration process.



the physical appearance of the device must be considered for possible BadUSB device. For example, if a flash drive is shown as a keyboard, the user must suspect that the device might be maliciously reprogrammed. Currently, the identification of BadUSB devices can be achieved only after the device is enumerated by the host.

### 3.1.3. Behavioral characteristics of BadUSB devices

At the time of enumeration, all of the intended features of the device must be declared for the host controller's approval. Therefore, a BadUSB cannot be effective until the intended feature is recognized and initialized by the host controller. If a device wants to add a feature that was not declared at the previous enumeration, it must re-enumerate. Therefore, we identify the enumeration stage to be protected for BadUSB attack.

Among many common device types such as network adapters, video devices, and human input devices (HIDs), a keyboard is easy to simulate because of its relatively low hardware requirements. Launching arbitrary commands with a currently logged-on user is a significant threat. Therefore, it is a suspicious event for the operating system when a keyboard is trying to enumerate when the plugged-in device is physically not.

Also, a potential BadUSB device might attempt to avoid detection by appearing as its original features initially, then re-enumerating at a later point of time. Although valid legitimate use cases exist, such as a cellular modem that itself contains drivers, we believe that it is still considered suspicious.

## 3.2. Design and operation of USBWall

In this section, we discuss the high-level design of the USBWall. USBWall operates on two hardware components. In addition to the host's USB controller, the BBB is connected to the host with two types of cables. We use CAT5e cable for the control channel, and a mini-USB cable for actual USB

traffic. Second, we discuss the operation of USBWall in detail. The BBB handles the enumeration of a newly connected USB device. The host then issues a `lsusb` command to inquire of all connected devices' enumeration details. The BBB relays the information to the host to be parsed and be shown via USBWall's user interface. We also show the benefits of enumerating a USB device on a separate device. Finally, in the expected protection section, we speculate about the realized benefit of USBWall, which enumerates USB devices in a sandboxed environment.

### 3.2.1. Design of USBWall

With the realization that the current USB specification places excessive trust when initializing a device, we propose a middleware solution to decrease the trust between the host and devices. USBWall utilizes two major components, the BBB (Coley, 2013) and the user interface (UI), which runs on host machine. The BBB is an open-source embedded computer that runs Debian 3.12.0-bone8 operating system. Powered by a 5-V, 2-A DC power supply, the BBB is connected to both the host computer and the suspected device. The BBB provides the hardware platform for USB connectivity between the host's USB controller and the BBB's. Figure 6 shows the components of USBWall.

USBWall requires a network connectivity between the host and the BBB. Via SSH, the USBWall UI issues and obtains the information to display to the user. The USBWall UI uses the SSH.NET library to handle the `lsusb -v` command to the BBB (Renci, 2014). The BBB, in turn, relays the enumeration details to the host. Once the host receives the details, USBWall parses the result of the `lsusb` into more readable format. It is important to note that the entire enumeration processes take place entirely on the BBB. The host's USB controller is unaware of the process until the user confirms via the UI, which sends a command over SSH. Although the implemented USBWall runs on



Figure 6. Diagram of USBWall's components.

Windows 7, because of the universal nature of the control channel (SSH), it is a platform-independent solution that can run on multiple systems.

### 3.3. Operation of USBWall

For proper operation of USBWall, it must be placed and operated between the host and the suspected USB devices. When a device in question is plugged into the BBB, the device goes through the enumeration process automatically. The BBB retains the information until requested by the USBWall UI from the host. When the user launches the UI, it inquires about the device's enumeration details to the BBB using `lsusb -v` via SSH. Once the USBWall's UI receives the device details from the BBB, the UI parses the result and displays it in a tree format. Figure 7 illustrates a sequence of message exchanges in the USBWall; Figure 8 shows a screenshot of USBWall's user interface.

Since the BBB's USB enumeration is isolated and independent of the host's USB controller, the device detail is obtained without the involvement of the host. If the suspected device is designed to send preset keystrokes, any attempt at keystrokes from the bogus device is not transmitted to the host. In other words, when the enumeration at the BBB is completed, the device will try sending keystrokes to the host into which the device is plugged. In this case, the BBB is the host. The BBB's local console, to which the device's input is directed, remains at the Linux standard log-in screen. Therefore, any keystroke attempts sent to the host are directed to the BBB instead, which ignores them unless the keystroke precisely matches the log-in information of a user who has shell access. This provides an additional protection layer for the middleware itself.

After receiving the enumeration details, the user checks whether the device matches the physical characteristics. To assist with the decision, the UI will color-code certain entries of a device's details.

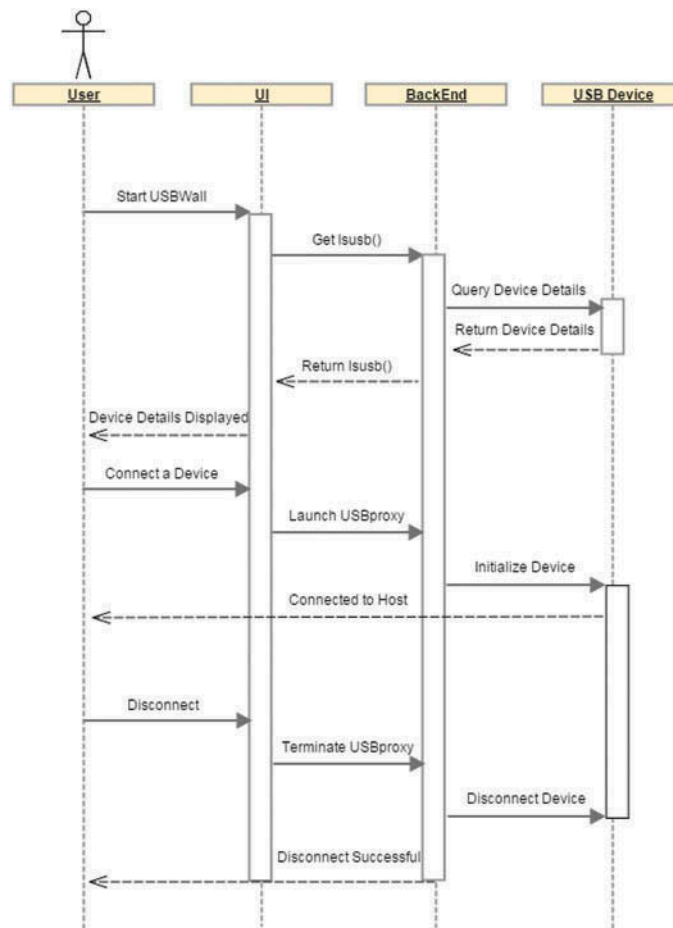


Figure 7. Sequence diagram of USBWall.

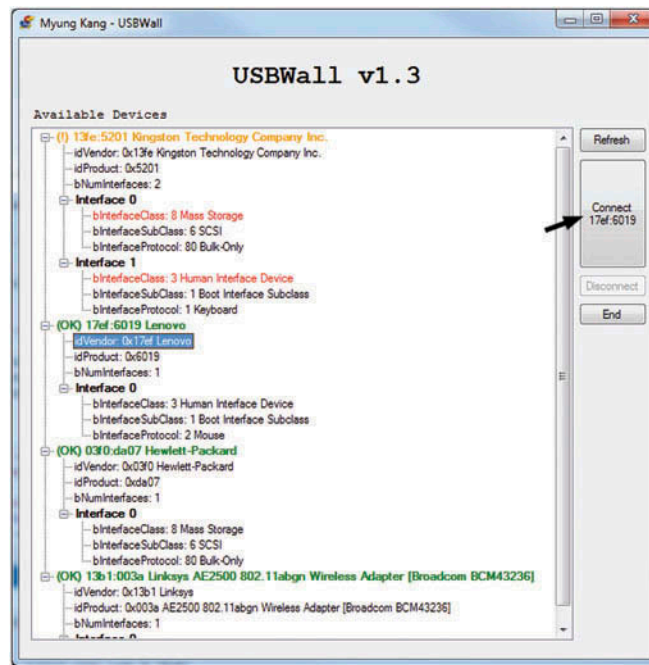


Figure 8. Middleware front end (UI) on Windows host.

The user is prompted by an option to choose which device to connect to the host and start relaying.

When a device is chosen and the user clicks the connect button, the UI issues the `sudo -S usb-mitm -v 'VID' -p 'PID' &` command to the BBB. VID and PID are the vendor ID and product ID of the device, respectively. `usb-mitm` is the filename of the binary of `USBProxy`. `USBProxy` (Spill & Stasiak, 2014), an open-source project by Dominic Spill, is used to emulate the function of the device on the BBB's host-facing interface using the `gadgetfs` Linux subsystem. `USBProxy` is launched with the PID and VID of the device to ensure that only the selected device is relayed. To emulate the device functionalities, `USBProxy` utilizes `gadgetFS`, which is part of the BBB's Debian 3.12.0-bone8 operating system.

### 3.4. Expected protection

Using the BBB's USB controller along with `USBProxy`, `USBWall` provides the user sufficient information to decide whether the device is safe to use. By issuing the `lsusb` command to the BBB via the SSH terminal, the UI obtains the information of the device plugged in before it has a chance to present to the host. `USBWall` parses and highlights parameters such as `bNumInterface` and `bInterfaceClass` to help

users assess the risk quickly. Those parameters are key items when assessing the likelihood of the device being malicious. Figure 9 shows the source code that issues the `lsusb -v` command to the BBB to obtain the USB device's information from the BBB to the host. The result is parsed and stored into a structure for later display.

By effectively separating the host and a suspected device while maintaining the user's ability to interact with the device, we expect that the separation will provide sufficient displacement of trust to block any devices with malicious firmware from launching an attack by posing as a different device than its physical form factor. The ability to assess the device's intention without exposing the host's USB controllers effectively protects the host from reprogrammed firmware attacks. The risks originating from unknown user devices is greatly reduced. Furthermore, the concept of `USBWall` that allows the protection without any substantial changes to the kernel makes `USBWall` an easy candidate to be ported to different operating systems.

## 4. Validation for sandboxed USB enumeration

This section discusses the results and evaluates of the effectiveness of `USBWall` against USB devices with

```

Private Function lsusb() 'returns lsusb in struct in strings
Dim templsusb As String = $shrunCmd("lsusb -v") 'get lsusb -v result from BBB
Dim arr_lsusb As str_lsusb() 'init struct
ReDim arr_lsusb(50) 'assume max 50 USB devices
Dim templsusbarr As String() = templsusb.Split(vbCrLf) 'split result by return carriage
Dim i As Integer = -1
Dim bIntNumFor As Integer = 0
For Each line As String In templsusbarr
    If line.Contains(": ID ") Then
        i = i + 1 'increment i at header of each device
        arr_lsusb(i).EasyDesc = line.Substring(23).Trim
    End If
    'we are interested in the following fields
    If line.Contains("idVendor") Then arr_lsusb(i).idVendor = line.Substring(22)
    If line.Contains("idProduct") Then arr_lsusb(i).idProduct = line.Substring(22)
    If line.Contains("bNumInterfaces") Then arr_lsusb(i).bNumInterfaces = CInt(line.Substring(20))
    If line.Contains("bInterfaceNumber") Then
        bIntNumFor = CInt(line.Substring(24))
        ReDim Preserve arr_lsusb(i).bInterface(bIntNumFor)
    End If
    If line.Contains("bInterfaceClass") Then
        arr_lsusb(i).bInterface(bIntNumFor).bInterfaceClass = line.Substring(31)
    End If
    If line.Contains("bInterfaceSubClass") Then
        arr_lsusb(i).bInterface(bIntNumFor).bInterfaceSubClass = line.Substring(31)
    End If
    If line.Contains("bInterfaceProtocol") Then
        arr_lsusb(i).bInterface(bIntNumFor).bInterfaceProtocol = line.Substring(30).Trim()
    End If
Next
ReDim Preserve arr_lsusb(i) 'when done, resize array before returning
Return arr_lsusb
End Function

```

**Figure 9.** USBWall's snippet of issuing and parsing lsusb.

malicious firmware. To simulate an attack scenario, BadUSB and BadAndroid devices are connected to a Windows 7 host computer with USBWall installed. First, we compare the protection efficiency of commercially available antivirus products with USBWall. The suspected devices are presented to a protected host by each commercially available antivirus program. We use a USB storage device with Psychson's HID payload applied to simulate a BadUSB to test whether the arbitrary code runs successfully. The HID payload is designed to launch a series of keystrokes that will cause a Windows machine to run the notepad application, then type predefined characters (SurfKahuna at hak5darren, n.d.). If a notepad opens without any user interaction upon plug-in, we consider it a successful attack. If the notepad does not open, and the user is notified of the risk, we consider it an unsuccessful attack as well as a successful protection. In each case, we verify whether the host is protected from BadUSB attack. Second, we will compare BBB's effect on the data throughput. Using a publicly available tool, CrystalDiskMark (Kashiwano, n.d.), we test 100 MB data transfer throughput for sequential, 512 KB, and 4 KB.

#### 4.1. Experiment environment

USBWall is developed in Linux, and the Windows user interface portion is written in Visual Studio 2012 Professional. Linux is used on the BBB,

which runs a Debian distribution to facilitate the middleware functionality. An open-source project, USBProxy, is also used to relay the USB data. On the host computer, the UI is developed using Visual Basic 2012 Professional. A Toshiba TransMemory, 16GB PFU016U-1BCK, is chosen as a test BadUSB device. The drive is applied with HID-emulating firmware using Harman's tool (Harman, 2014). The specification of the components used in testing is shown in Table 1.

We compare USBWall's protection with AVG, avast!, Windows Defender, and with no antivirus installed for control. (Avast, n.d.; AVG Technologies, n.d.; Microsoft, n.d.) The test firmware is written to launch a notepad using Rubber Ducky script (Hall, n.d.). If a notepad launches and keystrokes are entered without any user intervention, we consider the protection to be ineffective, as it allows the device to run arbitrary keystrokes. Figure 10 shows the notepad launched on the host with no antivirus when a test device is plugged in. Upon plugging in, the notepad is opened. Without any further user actions, keys are typed into the notepad. We use this scenario to determine whether each protection is effective against BadUSB attack.

File transfer throughput is also measured by performing benchmark tests with CrystalDiskMark (CDM) (Kashiwano, n.d.). CDM performs read and write operations in a predefined size of data at different sizes of blocks. We run tests five times and

**Table 1.** Specifications of components used in testing.

Type	Sub type	Specifications
<b>Host computer</b>	CPU	Intel Core2 Q9650 3.00 GHz
	RAM	8 GB
	Storage	1 TB
	Operating system	Windows 7 Professional 64-bit
	USB controller	Intel X48 Chipset with ICH9R
<b>USBWall (BBB)</b>	CPU	Sitara XAM3359AZCZ 1 GHz
	RAM	512 MB
	Storage	2GB eMMC
	Operating system	Debian 3.12.0-bone8
	Power supply	5V 2A (5.5 mm × 2.1 mm)
<b>Development tool</b>	User interface	Visual Studio 2012 Professional
<b>Sample USB devices</b>	Psychson-Applied	Toshiba TransMemory
	Simulated BadUSB drive	16GB PFU016U1BCK
	BadAndroid-v0.1 device	Samsung SPH-D700

average the result. To test the throughput when USBWall is in use, we first connect a USB device to the BBB. The BBB is connected to the host USB port via mini-USB cable. USBWall UI is launched on the host computer to initiate the transfer. CDM is launched with various data block size choices. While CDM is performing the test, we monitor the CPU usage on the BBB via `ps -ef` command on the BBB on Putty v0.60. CPU data is collected on the Windows 7 64-bit host computer without any antivirus program to ensure the integrity of results.

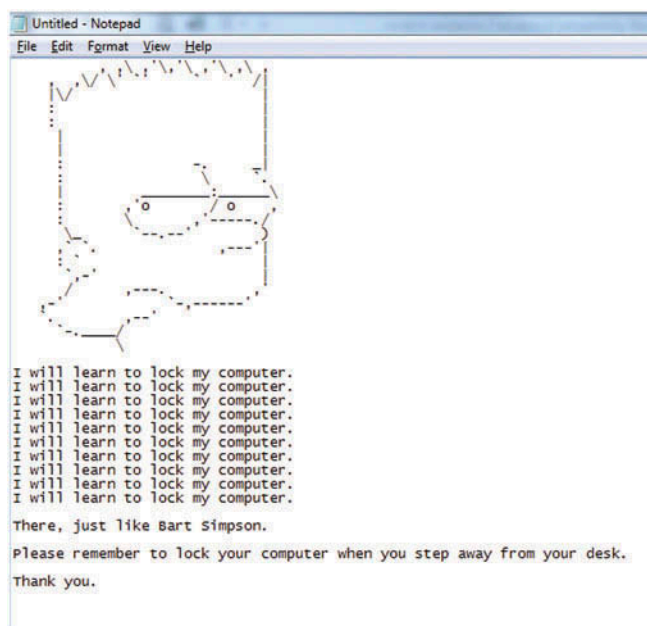
## 4.2. Experiment results

We test the efficacy of USBWall by comparing whether the sample BadUSB device successfully launches an attack on different host configurations with currently available antivirus solutions. If the test script runs, we conclude that the protection is ineffective. Later, we test the same sample device with USBWall.

To test and compare the effectiveness of different protections, we use a Psychson HID payload example (Harman, 2014). A BadUSB device is presented to hosts with different antivirus protections that are available commercially at the time of writing. Selected antivirus software include AVG Free, avast! Free, and Windows Defender. The test device is inserted into the host while running each antivirus program with full protection options. If the notepad opens and text is typed, we consider the protection ineffective against BadUSB attack, as it allows the BadUSB's firmware to launch its preset keystrokes. The results show that no commercially available protections detect or block the test BadUSB devices.

### Config 1: Control—No antivirus

As a control, the sample HID payload is plugged into a host without any type of virus



**Figure 10.** Screenshot of successful launch of HID payload launch.

protection installed. As expected, the sample malicious firmware runs on the host, opening a notepad and typing preset keystrokes. This result is compared to other test cases with antivirus software.

### Config 2: AVG by AVG Technologies

The host is equipped with AVG Free antivirus with the most recent database pattern at the time of writing (AVG Technologies, n.d.) (database version 4306/9296, AVG Antivirus FREE 2015.0.5751). Even with all real-time protection offered by the antivirus enabled, BadUSB is able to run without any user interaction. Figure 11 shows the successful launch of the firmware's preset texts with AVG running.

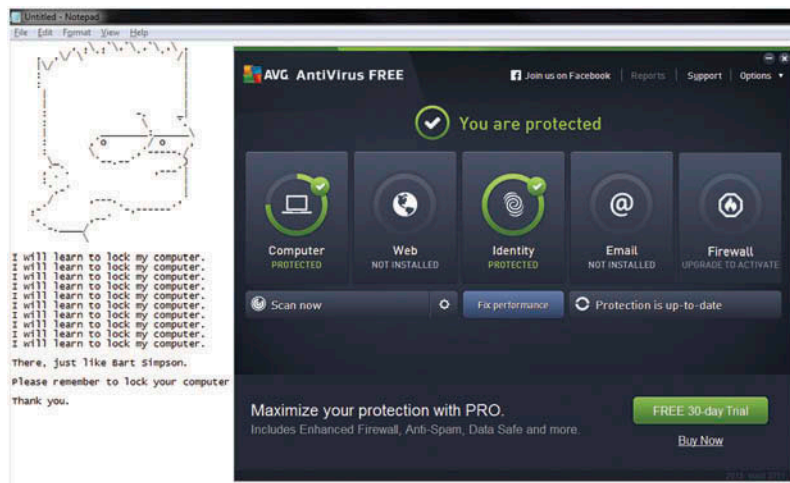


Figure 11. BadUSB HID payload launched with AVG antivirus.

### Config 3: Avast! by Avast

Next, the host is equipped with avast! antivirus software by Avast. The antivirus is allowed to update to the most recent database pattern at the time of writing (Avast, n.d.) (database version 150313-2, Avast Free Antivirus 2015.10.2.2214). The test BadUSB device is plugged in, and is able to run without any user interaction. Figure 12 shows the successful launch of the firmware's preset texts with avast! running.

### Config 4: Windows Defender by Microsoft

Last, the host is prepared with Windows Defender by Microsoft. The sample BadUSB device is plugged in after Windows Defender is updated to the most recent database pattern at the time of writing (database

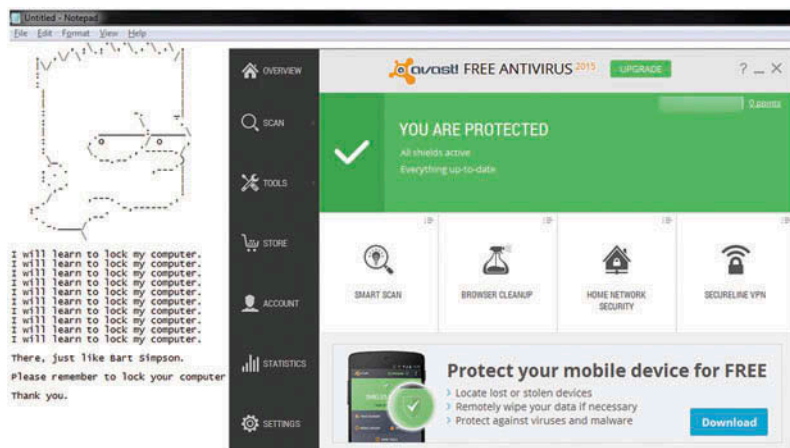


Figure 12. BadUSB HID payload launched with avast! antivirus.

Version of 1.193.2482.0, Windows Defender 6.1.7600.16385). Even with all real-time protection enabled, BadUSB is able to run without any user interaction. Figure 13 shows the successful launch of the firmware's preset texts with Windows Defender running.

### 4.3. BadUSB devices with USBWall

The previous tests show that the current antivirus approach is ineffective in preventing a BadUSB attack. This is due not only to the nature of the attack, which exploits the process during enumeration, but also to the access to the malicious firmware storage area, which is proprietary. Lacking a standard way to access the firmware storage area, it remains largely impossible for antivirus software to scan the firmware of devices. Instead, USBWall focuses on the key fact that the BadUSB's actions, whether legitimate or malicious, remain valid within USB specifications. In this section, we test USBWall's efficacy against two types of BadUSB devices. Psychson is the name of Harman's public project (Harman, 2014). Second, a proof-of-concept BadAndroid-v0.1 script published by SRL is presented to USBWall. In both cases, USBWall is effective in showing the user the detailed intention of each device while maintaining the host's USB controller isolated.

#### 4.3.1. Psychson devices with USBWall

Toshiba TransMemory 16 GB is a small flash drive that appears harmless. However, when the

Psychson-applied device with HID payload is connected to USBWall, the user can easily tell that the device presents itself as an HID keyboard. The user launches USBWall by double-clicking the icon on the desktop after plug-in of the suspected device. At start-up of USBWall, it shows the tree list of connected devices as shown in Figure 14, obtained by the BBB using an `lsusb -v` command.

As shown in Figure 14, it is not only possible, but also easy, to pinpoint the device's intent. USBWall highlights the entries of the suspected device's properties in different colors. USBWall highlights the detail of Kingston Technology Company's (VID 13FE and PID 5201) two interfaces. The device's intentions include a keyboard. Upon reading the result, the user immediately realizes that the device may not be presenting itself and suspects a BadUSB attack. As the USB handshake happens only at the protected level of the BBB, the user can disconnect the device safely.

Although we assume all devices as unsafe, we believe that the user must have a way to override the warning and initiate the device. Therefore, the user still has permission to proceed connecting the device if the user believes that it is a valid intent. The device is connected when the user gives an explicit permission using the UI by selecting the entry, then clicking the Connect button.

#### 4.3.2. Bad Android devices with USBWall

BadAndroid script made available by SRLabs turns an Android phone into a network adapter for man-in-the-middle (MITM) attack (Security

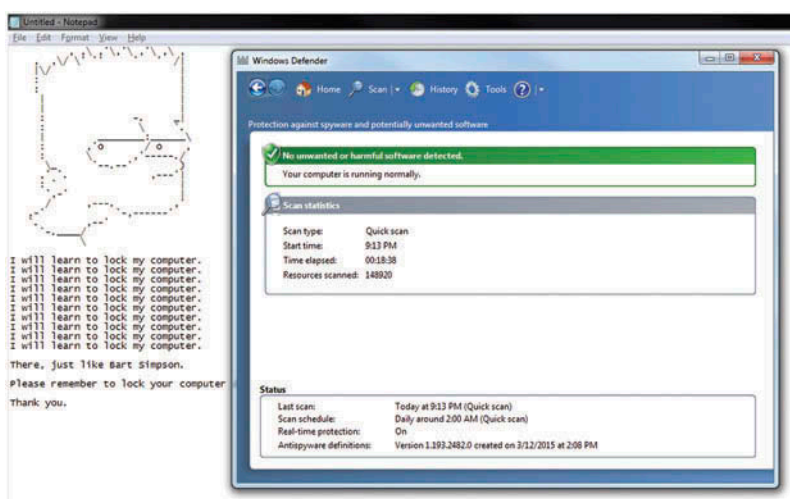


Figure 13. BadUSB HID payload launched with Windows Defender.

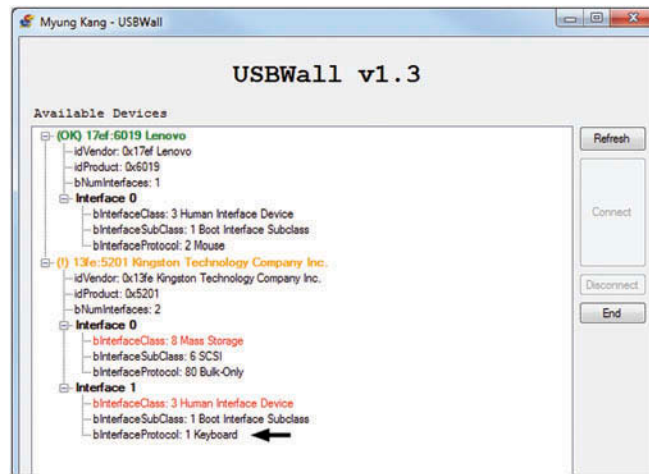


Figure 14. USBWall detection of Psychson-applied sample BadUSB device.

Research Labs, 2014). The script is launched on a Samsung SPH-D700 Android phone before connecting the micro-USB cable to the BBB. When the BadAndroid-enabled mobile device is presented to a Windows 7 host with no protection, Remote Network Driver Interface Specification (RNDIS) devices are automatically installed and initialized with no user interaction.

We test the same scenario with USBWall. When the suspected device is connected via USBWall, the UI shows the user the intention of the device. As with the previous test, the user launches USBWall by double-clicking the icon on the desktop. At start-up of USBWall, it shows the tree list of connected devices as shown in Figure 15 obtained by the BBB using `lsusb -v`. Now that the user is notified of the details of the device. The user has an option to connect the device to the host if it is a

valid device insertion. Figure 15 shows the detection of the device with VID of 04E8 and PID of 6863 is presenting itself as a RNDIS device.

As with the previous test, the user still has the right to proceed connecting the device despite the warning provided by USBWall. The user simply needs to select the device and click the Connect button. If the user decides not to use the device, the user can simply disconnect it without worrying about the safety of the host.

#### 4.4. Performance test

While USBWall protects the host and relays the USB data, it must introduce an overhead to the traffic path as it is an additional node. Therefore, we believe that it is worthwhile to measure the difference in the data throughput. If the overhead

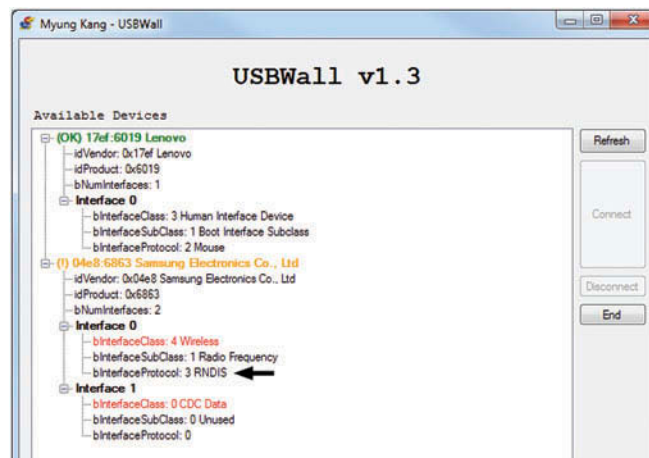


Figure 15. USBWall detection of BadAndroid-activated mobile phone.



**Table 2.** CrystalDiskMark result (read, 100 MB, MB/s).

Type	Test size	1	2	3	4	5	Average
Via USBWall	Sequential	1.975	1.977	1.977	1.950	1.952	1.966
	512 K	1.956	1.981	1.981	1.936	1.955	1.962
	4 K	1.421	1.449	1.419	1.428	1.433	1.430
Direct	Sequential	21.19	21.27	21.15	21.23	21.18	21.20
	512 K	21.28	21.35	21.23	21.24	21.13	21.25
	4 K	5.560	5.564	5.586	5.532	5.510	5.550

causes significant negative effect on the usability of the device, it would make USBWall less appealing to the user.

In this test, we use JD Secure II+ 2 GB by Lexar with CrystalDiskMark (CDM) to test the read and write performance. We test the read-and-write throughput using CDM with 100 MB data transfer. 100 MB data is transferred to the device in 512 KB, 4 KB, and sequentially with and without USBWall.

As shown in Tables 2 and 3, there are clear differences in transfer speed between a direct connection and via USBWall. We believe that the decreased performance is due to the overhead introduced to the data path, due mainly to the limited processor capability of the BBB, which is responsible for relaying the data. The copying of USB data traffic involves interacting with the device as well as relaying to the host at the same time. One process must wait for the other while the transaction is finished. To support the analysis, Figure 16 shows the CPU utilization on the BBB while data transfer test is in progress. It shows that the USBProxy process, usb-mitm, takes nearly 90% of the processor. We believe this is purely a technical limitation of the BBB and USBProxy. Hence, we believe that this will be resolved with a faster middleware platform. While it affects the speed of

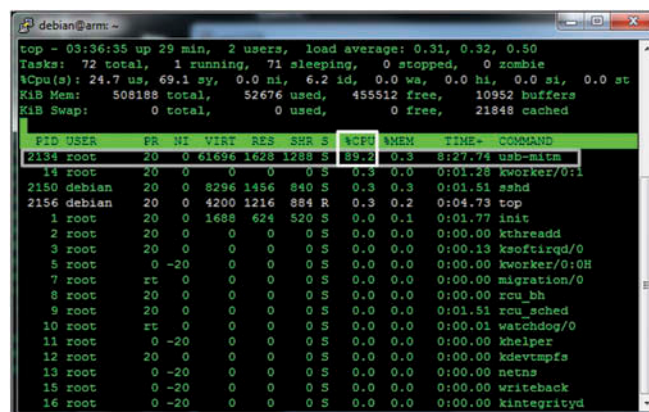
the data, it does not affect the integrity of the data transmitted.

#### 4.5. Validation conclusions

In this article, the implementation of USBWall has been explained, and tested against Psychson and BadAndroid devices. AVG Free, avast! Free, and Windows Defender have been tested against the sample BadUSB devices. We showed that no existing antivirus protections are capable of blocking or detecting the presence of the device with malicious firmware. We assess that the difficulty of existing approach comes from the fact that the malicious firmware resides in the firmware storage, which is generally inaccessible to antivirus solutions. Also, the accesses to firmware storage of a USB devices are mostly proprietary. Lacking a universal way to

**Table 3.** CrystalDiskMark result (write, 100 MB, MB/s).

Type	Test size	1	2	3	4	5	Average
Via	Sequential	1.890	1.886	1.889	1.907	1.906	1.896
USBWall	512 K	1.159	1.145	1.173	1.147	1.088	1.142
	4 K	0.023	0.022	0.023	0.023	0.022	0.023
Direct	Sequential	6.797	6.775	6.823	6.771	6.970	6.827
	512 K	2.140	2.122	1.947	2.034	2.093	2.067
	4 K	0.024	0.023	0.023	0.024	0.023	0.023

**Figure 16.** ps -ef of BBB while transferring.

access the firmware storage area, the antivirus programs have no way to distinguish whether the firmware's behavior is legitimate or malicious.

We have showed the effectiveness of USBWall with Psychson-applied HID sample BadUSB and BadAndroid-v0.1 devices by SRL. For both sample devices, USBWall successfully shows the user the intentions of the sample devices. As the USB handshake happens only at the protected level of the BBB, the user can disconnect the device safely without affecting the host's security.

While the conventional antivirus protections have been shown to be completely ineffective, USBWall successfully isolates the devices from the host and notifies the user of the intentions of connected devices. The test results are summarized in Table 4.

#### 4.6. Hardware and software considerations

USBWall utilizes BBB Rev. A5A to act as a gateway and a proxy to enumerate and relay the details of the connected devices. While it provides the host protection from any USB devices with malicious firmware, we find that the data transfer speed is slower than direct connection. Although the BBB has a 1-GHz Siatara XAM3359AZCZ processor (Coley, 2013), a performance degradation is inevitable when it comes to data throughput. This is because USBProxy (Spill & Stasiak, 2014) operates at a software layer utilizing gadgetFS as well as the inherent delays in the added nodes. We believe that the enhanced throughput is attainable with an upgraded BBB with faster hardware. Nevertheless, the integrity of transmitted data is not affected. Therefore, we believe USBWall's effectiveness against BadUSB devices remains valid. Although USBWall UI is currently developed only for Windows host with .NET framework 4.5, it can be easily ported to other operating systems thanks to the universal underlying protocol, SSH. To port USBWall to other

**Table 4.** Comparison of protections against sample BadUSB devices.

	AVG Free 2015.0.5751	avast! Free antivirus 2015.10.2.2214	Windows Defender 6.1.7600.16385	USBWall v1.3
Psychson- HID	No	<b>No</b>	No	<b>Yes</b>
BadAndroid	No	No	No	Yes

platforms, only the UI needs to be rewritten, because the back-end communication and USB enumeration is done on the BBB independently.

#### 4.7. Summary

In this section, we discussed the result of USBWall against USB devices with malicious firmware. The HID payload test sample and BadAndroid script were used to test the efficacy of each protection. Under various commercially available protections including AVG Free, avast!, and Windows Defender, the malicious firmwares executed successfully. We conclude that currently existing antivirus protects are inadequate to provide sufficient protection due to the nature of the USB standard. The lower-level transaction is manipulated by the firmware, which is not visible to the antivirus product. USBWall, however, successfully prohibits malicious firmware from launching by enumerating the device in a sandboxed environment. Through the protection environment, the host can safely retrieve the detailed information about the device to decide whether or not to accept the device. USBProxy is then used to bridge the connection and introduce the device to the host.

We also tested the file transfer throughput to measure the impact of having an additional node on the USB data path. After analyzing the CPU utilization on the BBB, we find that the BBB's processing power affects the performance negatively. However, the efficacy of the protection remains effective, as the data integrity is not affected. For USBWall protection on operating systems other than Windows, we believe the porting effort is minimal because only the user interface needs to be ported.

#### 5. Contributions and future work

The system discussed in this article protects computers from BadUSB devices by introducing a middleware, BBB, to relay the device enumeration information before the host is exposed to the device. Although many creative methods are used in the current design, there exist possibilities to enhance the system further. For example, due to the lack of serial-number enforcement by the current USB specification, it is impossible for the operating system kernel to distinguish one device from another. However, with the BBB's access to the hardware-

level information, it may be possible to fingerprint a device based on its electrical characteristics such as communication delays and power consumption patterns during the enumeration.

Second, USBProxy uses gadgetFS. After lsusb is displayed and the user decides to connect, USBProxy relays the data using the gadgetFS subsystem. Although it is capable of viewing the properties of all devices, if a type is not supported by the gadgetFS subsystem, it will fail to relay. As more device types are added to gadgetFS in the future, such as wireless dongles and other physically small devices, we anticipate wider device support in USBWall.

Third, consider the BBB's overhead affecting performance in Section 4.4. Not only is the performance restricted to USB 2.0, but also to the capability of the CPU, which decreases the performance further. As newer versions of BBB are released with faster hardware, we anticipate better performance.

Last, we strongly believe that the concept of introducing a middleware for physical computer peripherals is a powerful candidate for offering protection against user devices. Especially with the emerging risks from hardware trojans (Clark et al., 2011; Clark, 2009), the approach of USBWall can be extended to more interfaces than USB.

All the scenarios above are excellent candidates for work to be explored. By filling in the gaps of current restrictions, such as IEEE 1667 (IEEE 1667 Working Group, 2010), we firmly believe that it will contribute to safer computing from malicious user devices. We believe that where IEEE 1667 has failed to reach, USBWall will reach a larger user base thanks to the minimal changes required on operating systems.

## References

- Appavoo, J., Hui, K., Soules, C. A. N., Wisniewski, R. W., Da Silva, D. M., Krieger, O., ... Xenidis, J. (2003). Enabling autonomic behavior in systems software with hot swapping. *IBM Systems Journal*, 42(1), 60–76. doi:10.1147/sj.421.0060
- Arce, I. (2005). Bad peripherals. *Security & Privacy, IEEE*, 3(1), 70–73. doi:10.1109/MSP.2005.6
- Avast. (n.d.). *Avast | The World's #1 Antivirus Software*. Retrieved September 25, 2014, from <https://www.avast.com/index>
- AVG Technologies. (n.d.). *AVG free antivirus & malware protection*. Retrieved September 25, 2014, from <http://free.avg.com/us-en/free-antivirus-download>
- Brandom, R. (2014). *USB has a huge security problem that could take years to fix*. Retrieved November 18, 2014, from <http://www.theverge.com/2014/10/2/6896095/this-published-hack-could-be-the-beginning-of-the-end-for-usb>
- Clark, J. (2009). *On unintended USB communication channels* (Master's thesis). Royal Military College of Canada, Kingston, Canada.
- Clark, J., Leblanc, S., & Knight, S. (2011). Risks associated with USB hardware trojan devices used by insiders. In *Systems Conference (SysCon), 2011 IEEE International* (pp. 201–208). Montreal, Canada: IEEE.
- Coley, G. (2013). *BeagleBone black system reference manual Rev A5.6*. Dallas, TX: Texas Instruments.
- Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips. (2000, April). *Universal Serial Bus specification v2.0*. USB Implementers Forum, Inc. Retrieved from [www.usb.org](http://www.usb.org)
- Falliere, N., Murchu, L., & Chien, E. (2011). W32. Stuxnet Dossier. *White paper, Symantec Corp., Security Response*. Cupertino, CA: Symantec Corporation.
- Future Technology Devices International Limited (FTDI). (2009). *Simplified description of USB device enumeration* (Technical note) (Vol. 1.0, pp. 14). Glasgow, UK: Author.
- G Data. (n.d.). *How to Be Sicher from USB Attacks*. Retrieved September 24, 2014, from <https://www.gdatasoftware.com/en-usb-keyboard-guard>
- Hall, J. (n.d.). *Duck Toolkit v.2*. Retrieved September 25, 2014, from <http://www.ducktoolkit.com/Home.jsp>
- Harman, R. (2014, January). *Controlling USB flash drive controllers: Expose of hidden features*. Presented at ShmooCon 2014, Washington, DC.
- IEEE 1667 Working Group. (2010, March). IEEE standard for authentication in host attachments of transient storage devices. *IEEE Std 1667–2009 (Revision of IEEE Std 1667–2006)*, pp. 1–125.
- Intel Corporation and Microsoft Corporation. (1999, May). Legacy plug and play guidelines. *A Technical Reference for Legacy PCs and Peripherals for the Microsoft Windows Family of Operating Systems, 1.0*.
- Jin, Y. (2012). *Trusted integrated circuits* (Ph.D. dissertation). Yale University, New Haven, CT.
- Karabarbounis, L., & Neiman, B. (2013). *The global decline of the labor share*. Technical report. Cambridge, MA: National Bureau of Economic Research.
- Kashiwano, M. (n.d.). *Crystal dew world*. Retrieved September 25, 2014, from <http://crystalmark.info/?lang=en>
- Mamiit, A. (2014). *How Bad Is BadUSB? Security experts say there is no quick fix*. Retrieved November 18, 2014, from <http://www.techtimes.com/articles/17078/20141004/how-bad-is-badusb-security-experts-say-there-is-no-quick-fix.htm>
- Microsoft. (2008). *Introducing enhanced storage access*. Retrieved November 18, 2014, from [https://technet.microsoft.com/en-us/library/Dd560657\(v=WS.10\).aspx](https://technet.microsoft.com/en-us/library/Dd560657(v=WS.10).aspx)
- Microsoft. (n.d.). *Microsoft security essentials - Microsoft Windows*. Retrieved September 25, 2014, from <http://win>

- [dows.microsoft.com/en-us/windows/security-essentials-download](https://www.microsoft.com/en-us/windows/security-essentials-download)
- Mitra, R. (2013). The information technology and business process outsourcing industry: Diversity and challenges in Asia. *Asian Development Bank Economics Working Paper Series*, 365(365), 4.
- Oshri, I., Kotlarsky, J., & Willcocks, L. (2015). *The handbook of global outsourcing and offshoring* (3rd ed.). London, UK: Palgrave Macmillan.
- PCI Security Standards Council. (2015, May). *Payment Card Industry (PCI) data security standard: Requirements and security assessment procedures, Version 3.1*. Wakefield, MA: PCI Security Standards Council, LLC.
- Renci. (n.d.). *SSH.NET library*. Retrieved September 25, 2014, from <https://sshnet.codeplex.com/>
- Rich, D. (2007). Authentication in transient storage device attachments. *Computer*, 40(4), 102–104. doi:10.1109/MC.2007.116
- Rueter, C. (2011). *The cybersecurity dilemma* (Ph.D. thesis). Duke University, Chapel Hill, NC, USA.
- Security Research Labs. (2014). *Turning USB peripherals into BadUSB*. Retrieved March 28, 2017, from <https://srlabs.de/bites/usb-peripherals-turn/>
- Spector, L. (2014). *BadUSB: What you can do about undetectable malware on a flash drive*. Retrieved November 18, 2014, from <http://www.pcworld.com/article/2840905/badusb-what-you-can-do-about-undetectable-malware-on-a-flash-drive.html>
- Spill, D., & Stasiak, A. (2014, January). *An open and affordable USB man in the middle device*. Presented at ShmooCon 2014, Washington, DC.
- SurfKahuna at hak5darren. (n.d.). *Payload lock your computer message*. Retrieved September 25, 2014, from <https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payload—lock-your-computer-message>
- Tetmeyer, A., & Saiedian, H. (2010). Security threats and mitigating risk for USB devices. *Technology and Society Magazine, IEEE*, 29(4), 44–49. doi:10.1109/MTS.2010.939228
- Usher, A. (2005). *Pod slurping*. Retrieved September 25, 2014, from [http://www.sharp-ideas.net/pod\\_slurping.php](http://www.sharp-ideas.net/pod_slurping.php)
- Verma, S., & Singh, A. (2012). Data theft prevention & endpoint protection from unauthorized USB devices. In *Advanced Computing (ICoAC), 2012 Fourth International Conference on* (pp. 1–4). Chennai, India: IEEE.
- Zhaohui, W., Johnson, R., & Stavrou, A. (2012). Attestation & authentication for USB communications. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on* (pp. 43–44). Gaithersburg, MD: IEEE.

## Biographies

**Myung Kang** completed his Master's of Science in Information Technology in 2015 and is currently an IT professional at City of Independence Power & Light, in Missouri.

**Hossein Saiedian** received his Ph.D. from Kansas State University in 1989. He is currently the director of IT degree programs, an associate chair, and a professor of computer science at the Department of Electrical Engineering and Computer Science at the University of Kansas (KU) and a member of the KU Information and Telecommunication Technology Center (ITTC). His research includes over 170 publications in a variety of topics in computer science, software engineering, and information security and his work in the past has been supported by the NSF as well as other national and regional foundations.