USBWall: A Novel Security Mechanism to Protect Against Maliciously Reprogrammed USB Devices

Myung-gu Kang

M.S., Computer Science, University of Kansas, 2015

Submitted to the graduate degree program in Electrical Engineering & Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Hossein Saiedian, Ph.D.,
Professor and Thesis Advisor

Bo Luo, Ph.D.,
Professor

Fengjun Li, Ph.D.,
Professor

Date Defended

The Thesis Committee for Myung-gu Kang certifies that this is the approved version of the following thesis:

USBWall: A Novel Security Mechanism to Protect Against Maliciously Reprogrammed USB Devices

Hossein Saiedian, Ph.D.
Professor and Thesis Adviser

Date Approved

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 which 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 function 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 it is stored and the way it behaves. Therefore, it is impossible for current file-level antivirus to be aware of such risk. Currently, there is no commercially available protection for USB devices other than mass storage devices. We propose a novel way to protect the host via a software/hardware solution we named a USBWall. USBWall uses BeagleBoard 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.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Hossein Saiedian, for his guidance, caring, discipline, and patiently correcting my writing throughout the research. Thank you for giving me the freedom to pursue the subject of my interest. I would also like to thank my committee Dr. Bo Luo and Dr. Fengjun Li for their valuable time and patience.

I would like to thank Mr. Dominic Spill and Mr. Karsten Nohl, who were more than willing to discuss about my questions and the ideas of their findings during their busy work hours.

This project would have been impossible without the support of Ellis Foundation and my employer Ward/Kraft, Inc. I would like to thank Mr. Roger Kraft for his generous support and genuinely believing in me. I would like to thank Mrs. Dana Ruhl, who has supported me throughout my thesis with her patience and understanding while I often miss work for research.

I would also like to thank my parents Sukchang Kang and Myungsoon Song, for their unending love and encouragement with their best wishes.

Finally, I would like to thank my lovely wife, Taewon Kim, and my daughter, Sophia Kang. Thank you. Thank you for always being there to cheer me up at hard times and allowing me time away to research and write.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement: 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 which allows it to be activated without additional software, the flash drive became the most popular USB device [1] [16].

To provide plug-and-play, the underlying protocol is designed to minimize user interaction in dynamically allocating system resources [9]. 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 user's inaccurate models of threat possibilities that make users believe they are safer than they actually are [33], a new threat appeared. The threat, coined as BadUSB by Security Research Lab, exploits the trusting nature of the current USB

specification to allow a device present itself as a different, potentially malicious, type of device [15] [20].

For example, a flash drive may establish the USB handshake as a keyboard. Exploiting user's inaccurately perceived sense of safety, seemingly small USB drive would appear more safer device than a physical keyboard. Being granted of 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 [20]. This is 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. As 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 port. When exploited, it can cause unintended behaviors ranging from annoying to compromised security measures.

This discovery differs from traditional security vulnerabilities like Stuxnet, which used USB drives as one of the primary means to propagate [10]. 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 antivirus can detect the existence of malicious firmware. Since 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 [36] and leaking confidential confirmation using unin-

tended channels [6] [7]. Due to 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 [33].

This thesis focuses on developing countermeasures against the 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 plugin events. Therefore, we developed USBWall as a novel way to add a protection layer with an additional middleware between the host and devices.

1.2 Significance of Sandboxed USB Transaction

In the world of cybersecurity arm race that is growing rapidly [29], our research explores a practical method to counter the newly found threat of USB devices with malicious firmware. Due to the nature of the vulnerability which operates in 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 the 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 device (Transient Storage Device, TSD) are well-discussed. In this section, we discuss the existing studies about the user devices' risk as well as proposed standards which attempted to make USB interfacing safer by authorizing and authenticating USB devices. Several proposals were submitted and approved. However, they were not implemented in the mainstream operating systems to offer effective protection for

users.

As more manufacturing is outsourced to other countries [24], the risk exists for an unauthorized change to be made to the product design at manufacturing phase. Even if the product meets all specification requirements, it has a possibility to operate unexpectedly under certain predefined circumstances. For instance, an encryption module which are mass-produced in distant factory can carry risks such as allowing unauthorized parties to access the private key on the chip's memory [17].

The term hardware trojan refers to an unauthorized change in hardware components to allow bypassing or weakening of designed behavior which are manufactured offshore [18]. A device with one or more components which could cause unexpected behavior. Increasing number of outsourcing of device fabrication foundries contribute to the heightened possibilities of malicious circuits inserted [25] [17]. 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 encryption module, returning false result, or a complete destruction of the module itself.

In addition to hot-plug capability and the smaller form factors, USB flash drives have quickly become the popular choice of removable storage device. While it provides users portability and convenience, the incorrect user-perceived notions of security has allowed a worm like Stuxnet to propagate and even reach stations not connected to the network [33] [10]. Reacting to such viruses like Stuxnet and its variants, endpoint security programs scan files before reading and writing. This had been proven its effectiveness in providing protection to operating system from malicious codes which attempt to execute without user’s consent.

The user is the most important entity because they are ones who have the phys-

ical access to the protected systems. The inaccurate idea of a device by a user, misjudged by its physical appearance, often leads to security threats [33]. For example, an iPod may be seen as a music player which needs charging using corporate computer. However, when it connects, it attempts to establish connections to the computer, which could lead to company policy violation. 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 notion and general notion of hardware trojan, unintended channels serve to extract data using unusual devices such as keyboard and USB speakers. Unintended channels utilize the data paths which are designed to carry control data for the devices [7] [6]. 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, a modified keyboard collects and stores data by tracking the LED state of Num Lock, Caps Lock, and Scroll Lock turn on and off. Although visible, this data path is generally not monitored by security softwares. Unsuspicious device like a USB speaker can also be used to extract data. By utilizing non-audible data channel such as `WAVEFORMATEXTENSIBLE`, it is also 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 of plugging in their USB device to charge could be just as risky as plugging in a flash drive with Stuxnet. Similar to the incorrect model of security in 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 public in 2005 [34]. The concept of Pod Slurping effectively proves that an innocuous device such as a music player can have a malicious intent programed so that the attacker can access the data on a protected system. Leveraging on the inaccurate model of security notion [33], consented use of user devices pose great threat to computing safety.

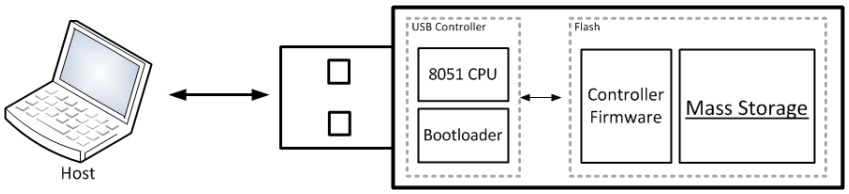


Figure 1.1: Diagram of USB Flash Drive Components

We propose a novel way to sandbox USB enumeration which are the root cause of the vulnerability on which the malicious firmware attack depend. The attack exploits the nature of USB devices as shown in Figure 1.1. Only mass storage area is visible to the user and the operating system. The attack executes at the firmware level, 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.

1.3 Research Methodology

The threats arising from maliciously reprogrammed USB devices are possible because of unnecessary and excessive trusts placed on devices by host, as well as the automatic install of common device drivers. We believe that reducing the implied trust is the key to protecting the host. To test the 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 automate several operating system's 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 1.2 shows how USBProxy [31] operates on BeagleBone Black (BBB), which is one of the key parts of the USBWall. A crucial part of the USBWall which handles the actual data is USBProxy by Dominic Spill. USBProxy relays USB data traffic from a device to host using `gadgetFS`. USBProxy is launched with parameter of the device's VID and PID. Therefore, if the device attempts to re-enumerate, USBProxy needs to relaunch. The flow is terminated automatically.

A test BadUSB-like device, the current project developed a BadUSB device by using Harman's program [15]. The device will have a different firmware from the manufacturer's. It will be presented to a host in several configurations. Test result will be gathered by comparing the product, USBWall, with other commercially available antivirus products.

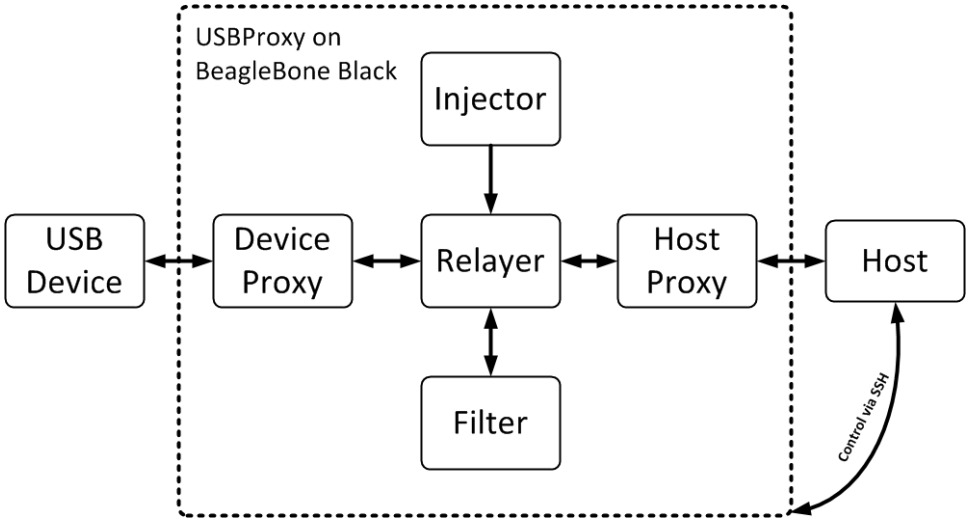


Figure 1.2: USBProxy Architecture [20]

1.4 Validation and Evaluation

The success of BadUSB attack hinges on the automatic enumeration and trusting its result to initialize the device. To lessen the implicit trust by automatic enumeration placed between host and device without significant operating system modification, we develop a program called USBWall using Visual Basic .NET for protected host. USBWall issues `lsusb -v` command to BeagleBone Black (BBB) middleware to obtain enumeration results. The results are transmitted back to the host via SSH session. The transmitted result is parsed by USBWall on the host to allow user to assess the intention of the device. We believe having device enumeration results beforehand eliminates the vulnerabilities BadUSB relies on to launch a successful attack. It integrates steps to monitor USB devices which are connected to the middleware (BBB).

The efficacy of USBWall is tested by measuring the rate of execution of a sample BadUSB device with various commercially available antivirus software. We use Psychson's HID Payload sample to emulate a BadUSB device. [15] Once the device is plugged in, we observe if the device's preset keystrokes are typed. We determine the protection insufficient if it runs. We test three major antivirus which are freely available at the time of writing. We observe that only USBWall alerts the user with yellow and red highlighted entries when a BadUSB device is connected. In addition, we test the effect of introducing an additional node in USB data path. We test the data throughput using CrystalDiskMark with 100 MB data transfer for sequential, 512 KB, and 4 KB. We use the average value of five test results for better confidence.

1.5 Thesis Organization

The rest of the thesis is organized as follow. Chapter 2 discusses related work in authenticated USB uses. USB standard is briefly explained to establish the inherent trust which exists in communication between the host and the device. We argue that the amount of trust is excessive and needs to be displaced. In addition, the chapter discusses the existing measures including non-technical, software, and hardware measures that can potentially hinder the malicious firmware attack. Based on the analysis, we attempt to theorize ways to deter the attack.

Chapter 3 discusses the design of the proposed solution and the expected protection for the host. We discuss the ways to identify BadUSB devices, and its behavioral characteristics. The process of USB enumeration is discussed to pinpoint the phase at which a BadUSB attack occurs. we reiterate that the need exists to reduce the amount of trust in USB interfacing. USBWall integrates series of shell commands and open source projects to achieve the untrusted environment while maintaining the ability enumerate the device. We expect that the risk is greatly reduced thanks to the sandboxed enumeration.

Chapter 4 validates the efficacy of USBWall. We expose USBWall and other commercially available antivirus products to two sample BadUSB devices. The result is collected on whether each protection is effective against two BadUSB devices. Additionally, as USBWall is an additional node introduced on USB data path, we test the data performance via USBWall to assess if there is any decrease in throughput while using USBWall.

Chapter 5 discusses about the limitations of the current version of USBWall and potential improvement for even broader protection. We summarize the contribution USBWall brings into the computing community for a safer USB interfacing. Future

works that could be done in the extension of our work is also discussed.

Chapter 2

Related Work in Authenticated USB Uses

Attack scenarios originating from user devices are well-known. From floppy disks, iPods, to USB devices, the full potentials of computer peripherals are also well recognized by security professionals while most ordinary computer users are unaware [2]. Historically, user devices have always carried risks of executing unexpected behavior to 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 was merely a medium on which to hold bits are now small reprogrammable computers with interfacing capability to its embedded storage flash chip.

2.1 The Inherent Trust of USB Standard

USB specification is written with the intention of minimizing the user intervention during the device initialization [9]. Consequently, the complexity of hardware required to be USB-compliant electronics became more sophisticated than older interfaces such as DB-9 and DB-25. While the handshake transactions of older interfaces

are handled at the application level, USB specifications mandate the host controller and slave controller to establish the initial contact using a predefined series of electronic signals handled at the hardware level. More details about USB enumeration are discussed in chapter 3. In addition to allowing for easier device initialization, to support 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 host side to determine the type of connected devices. Host controller initializes the device to the type of device solely based on the information received from the device.

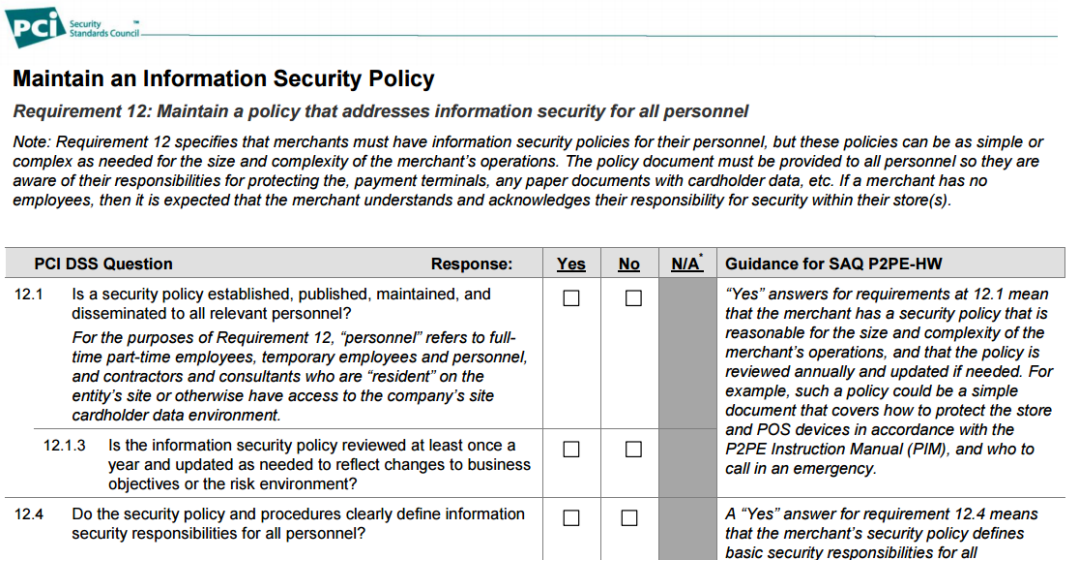
Unfortunately, the current USB specification implies the trust between host and device by not specifying a way to attest or authenticate the device. We argue that the properly placed trust is imperative to achieve safer computing. There exist several existing standards which can affect the plausibility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

2.2 Non-technical Measures

2.2.1 Policy Enforcement

A very limited number of countermeasures that mitigate this newly found threat exists. However, there are non-technical controls such as the payment card industry data security standard (PCI DSS) which could help reduce the exposure from devices with malicious firmware by mandating certain physical controls. PCI DSS disallows mostly all external computer interface except when it is absolutely necessary and no other alternative interfacing is available. In such case, compensating control must be declared and approved [26]. Figure 2.1 shows the example of PCI DSS's

self-assessment questionnaire mandating the policy reviews.



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? *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.*	☐	☐		*"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?	☐	☐		
12.4	Do the security policy and procedures clearly define information security responsibilities for all personnel?	☐	☐		*A "Yes" answer for requirement 12.4 means that the merchant's security policy defines basic security responsibilities for all*

Figure 2.1: PCI DSS Self-Assessment Questionnaires v2.0 Sample

Furthermore, employees handbook and acceptable use policy are also used to limit the 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 measure that prevent such attack to execute, although they provide sound legal ground to assist with legal proceeding.

2.2.2 Public Awareness

The possibility of the exploitation from USB devices with malicious firmware is announced at Black Hat USA 2014. At Security Research Labs (SRL)'s 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 a malicious firmware. The devices with malicious firmware, named a BadUSB, is

shown to appear as a completely different type of device than its physical characteristics. They did not disclose the details of how their demonstration BadUSBs were created at the time. Notwithstanding such gesture, the problem remained unattended.

Later that year, Harman made the code to reprogram the firmwares of certain types of USB flash drives available to the public at SchmooCon 2014 [15]. He confidently announced that his action hopes to bring manufacturers to fix the vulnerability quicker. He believes that the public awareness is the key to resolving the issue. Although this approach does not provide a technical means to protect against BadUSB, Harman's presentation made many users aware of the possibilities. These announcements were covered by several news outlets, alerting more general audience to be aware of the risk [5, 21, 30].

2.2.3 Proposed Standards to Secure USB Standard: IEEE 1667

To mitigate the risks from TSDs as well as unintended channels, a number standards and modifications of USB were proposed to attempt to authenticate and authorize a device [13, 28, 35, 36]. Authentication in USB means host and device can validate each other, and authorization means host only accepts a pre-defined functionality from devices. Current practice of using vendor ID (VID) and product ID (PID) only provide limited means of protation as they are easy to spoof, and only provide 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 only accepts pre-authenticated devices. As such, IEEE 1667 was proposed and approved in 2007 [13]. 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 does not seem to be any IEEE 1667-compliant devices in the general market.

2.3 Software Measures

2.3.1 G Data Keyboard Guard

Shortly after the discovery of the risk of devices with malicious firmware at Black Hat USA 2014 [20], G DATA published a program that traps USB keyboard insert events [12]. The program monitors all USB insert event, then the program is activated if the new device inserted is a HID keyboard. A pop-up notification as shown in Figure 2.2 appears for the user to decide. Depending on the choice, the insert

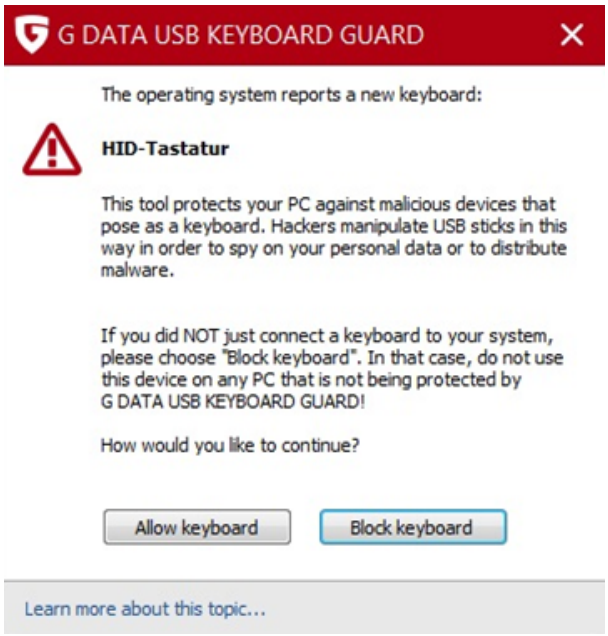


Figure 2.2: G DATA USB Keyboard Guard Notification

event is either allowed or blocked by the console user. If allowed, the new hardware is uninterrupted and result in a successful device initialization. If not allowed, the

new keyboard is not initialized and blocked for future insert. While the Keyboard Guard offers an excellent protection without any modification to Windows operating system, its protection is limited to HID keyboards inserted to Windows only. Considering that reprogrammed USB can pose as any device to any operating system, it does not offer a full protection for users against devices with malicious firmware posing other than a keyboard.

2.3.2 Enhanced Storage Access by Microsoft

The implementation of IEEE 1667 standard manifested in certain Windows product under the name of Enhanced Storage Access [22]. 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 manufactures.

2.4 Hardware Measures

2.4.1 Lack of Hardware Protection Against USB Devices with Malicious Firmware

Unfortunately, extensive research in databases of ACM and IEEE Xplore digital library yield no result of hardware protection against USB devices with malicious firmware. We believe that it is due to the nature of the vulnerability which executes at the low level layer of USB specification. Also, checking the validity of the USB device firmware might be overlooked by the security community.

2.5 USB: Time to Untrust

There still exists a need for a technical solution which protects from all types of devices with malicious firmware. Focusing on the layer and stage of the USB handshake at which the BadUSB exploit it, 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 the enumeration until it is deemed safe.

Therefore, 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 BeagleBoard Black (BBB) [8], a low-cost open-source computer, to act as middleware to enumerate the devices on behalf of the host. Like G Data Keyboard Guard [12], 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 the inconvenience 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 to use it. While similar protocols have been proposed [13] [36], they require significant changes within the operating system and the devices. The solution discussed in this thesis requires no changes to the operating system or the device.

2.6 Summary

In this chapter, we discussed about BadUSB devices' difference from previously known threats from user devices. As the bad code resides in the firmware area, rather than a storage area which are accessible, it is difficult to have a universal way

to scan to confirm the validity of the firmware. Thankfully, the findings were made public by insightful researchers. It is now up to the security community to come up with a good way to detect BadUSB attacks.

By exploiting the excessive implicit trust placed between host and USB devices, BadUSB can launch an attack by posing as a different device. A device which user would not expect from its physical appearance. The current USB specification is designed to accept the enumeration parameters provided by the device without verifying it. Therefore, this vulnerability remains largely unsolved.

There are, however, a few countermeasures that are applicable to the findings. PCI DSS standards, corporate acceptable use policy, and physical control can be used to increase the difficulty of BadUSB's success rate. PCI DSS offers some protection to limit the insertion of unauthorized devices by mandating to include such clause in company policy. Also, organization's acceptable use policy tries to enforce similar restriction. However, these policies do not offer technical protections to deter the attack from executing. As they are non-technical measures, they do not provide a proactive protection. At SchmooCon 2014, the tool to make a BadUSB device became available to public. By doing so, Harman aims to raise the public awareness, and force the manufacturers to improve their products [15]. IEEE 1667 is also standardized in 2007 [13], however, it remains largely unimplemented.

A very limited number of software and hardware countermeasures exist. Keyboard Guard by G Data protects Windows host operating system from a HID keyboard attack. Keyboard Guard monitors new device event and get user confirmation if it is a keyboard. While this is effective in protecting from a keyboard, it does not extend its protection outside of keyboard and Windows operating system. Enhanced Storage Access by Microsoft supports IEEE 1667-compliant devices to mitigate the issue completely. However, IEEE 1667-compliant devices are not found in the mar-

ket. After extensive research in several sources, we concluded that the hardware measures are non-existent due to the level of layer at which the attack happens. With hardware countermeasure lacking completely, we argue that the existing work is inadequate to provide safe and reliable USB interfacing.

Therefore, we suggest that the amount of trust must be lessened. By not trusting any devices, we believe that host is protected from a device with malicious firmware. Although similar standards have been proposed, they were hardly implemented. However, thanks to the USB specification which requires device to advertise its capabilities in order to be effective, we believe that monitoring the result of enumeration process will provide users a chance to block it from launching its attack. To aid the security threat from user devices, we propose USBWall. A way to identify a device's intention before connecting, without any changes on the operating system.

Chapter 3

USBWall: An Effective Middleware Protection

USB devices infected with malicious firmware, BadUSB, can only be detected by its discrepant intention at the enumeration stage. Due to the inherent risk user devices carry, several non-technical measures already exist that 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 public of the risks. With a precise understanding of the key stage of which BadUSB takes advantage, offering the user a chance to verify about the suspicious device can effectively prevent the BadUSB attack. As the device initialization can never take place on the protected host computer, the malicious firmware is not executed. This chapter discusses about USBWall in detail, a middleware solution that offers technical protection while maintaining the usability of USB interface.

3.1 Identification and Characteristics of BadUSB Devices

As more research on USB controller's potential capability progresses, researchers found that 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 a different type of devices 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 trojan as more manufacturing is outsourced [17], this poses a great threat to the users and render current anti-virus approach entirely useless because the infected firmware storage is inaccessible. The transactions which occur, albeit reprogrammed, remain legitimate in USB specifications. Therefore, operating system is unable to tell the difference between physical plugin and re-enumerating, and will attempt to accommodate the device a matching driver. The threat discussed in this thesis was first publicized by Security Research Labs [20]. At Black Hat USA 2014, Nohl demonstrated that masquerading as a different device was possible [20]. The details of how the device was reprogrammed was not released at the time. Nohl suggests the firmware signing to prevent unauthorized change. However, this approach will render the device unusable upon unauthorized change of the code. In this chapter, we discuss possible methods to determine if 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 [11]. Figure 3.1 shows the order of events which happens

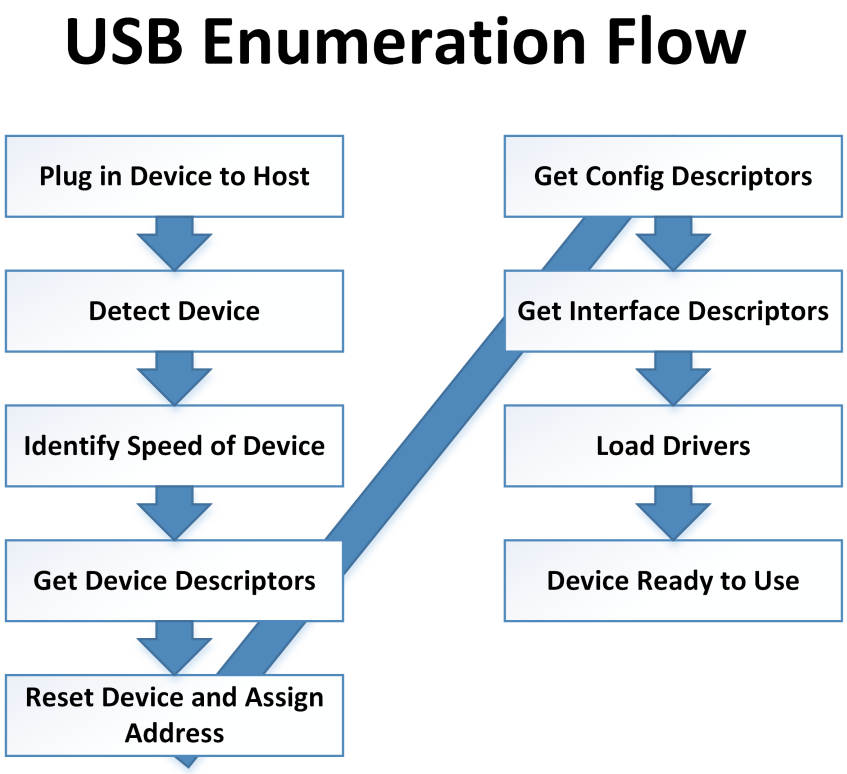


Figure 3.1: USB Enumeration Process

during the handshake. 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 case, the device is initialized at that point with no user interaction. The fact that a device can have more than one feature (interface), and they 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 a 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 with user's explicit consent.

3.1.2 Identification of BadUSB Devices

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

Identification of a BadUSB device may be possible if an unexpected device behavior is observed after inserting. Any behaviors inconsistent with 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 device is presenting itself to the host as a keyboard. We must suspect whether the device might be maliciously reprogrammed. Currently, the identification of BadUSB devices can only be achieved 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 host controller's approval. Therefore, BadUSB cannot be effective until intended feature is recognized and initialized by host controller. If a device wants to add a feature which was not declared at the previous enumeration, it must reenumerate. As such, we identify the enumeration stage to be protected for BadUSB attack.

Among many example interface such as network adapter, video device, and human input device (HID), a keyboard is easy to simulate because of its relatively low hardware requirements. Launching arbitrary commands with currently logged on user is a significant threat. Therefore, it is a suspicious event for the operating sys-

tem 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 reenumerate at a later point of time. Although valid legitimate use cases exist, such as a cellular modem which contains drivers in itself, 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, BBB is connected to the host with two types of cables. We use CAT5e cable for control channel, and a mini-USB cable for actual USB traffic. Second, we discuss the operation of USBWall in detail. BBB handles the enumeration of a newly connected USB devices. The host, then, issues a `lsusb` command to inquire of all connected devices' enumeration details. BBB relays the information to the host to be parsed and be shown via USBWall's UI. We also show the benefits of enumerating a USB device on a separate device. Finally, in expected protection section, we speculate the realized benefit of USBWall by enumerating 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, BeagleBone Black (BBB) [8] and User Interface (UI) which runs on host machine. BBB is an open-

source embedded computer that runs Debian 3.12.0-bone8 operating system. Powered by 5V 2A DC power supply, BBB is connected to both the host computer and the suspected device. BBB provides the hardware platform for USB connectivity between the host's USB controller and BBB's.

Figure 3.2: Diagram of USBWall's Components

USBWall requires a network connectivity between the host and BBB. Since the control session is via SSH, BBB must be reachable via TCP port 22. Via SSH, USBWall UI issues and obtains the information of the suspected device to display to the user. USBWall UI uses the SSH.NET library to handle `lsusb -v` command to BBB [27]. BBB, in turn, relays the enumeration details to host. Once the host receives the details, USBWall will parse the result of `lsusb` into more readable format. Example structure of `lsusb -v` is in Appendix 4. It is important to note that the entire enumeration processes take place solely on 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 Windows 7, because of the universal nature of the control channel (SSH), it is a platform-independent solution which can run on multiple systems.

3.2.2 Operations 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 BBB, the

device automatically goes through enumeration process. BBB retains the information until requested by USBWall UI from the host. When the user launches the UI, it inquires about the device's enumeration details to BBB using `lsusb -v` via SSH. Once USBWall's UI receives the device details from BBB, the UI parses the result, and displays it in a tree format. Figure 3.4 shows a screenshot of the USBWall's user interface.

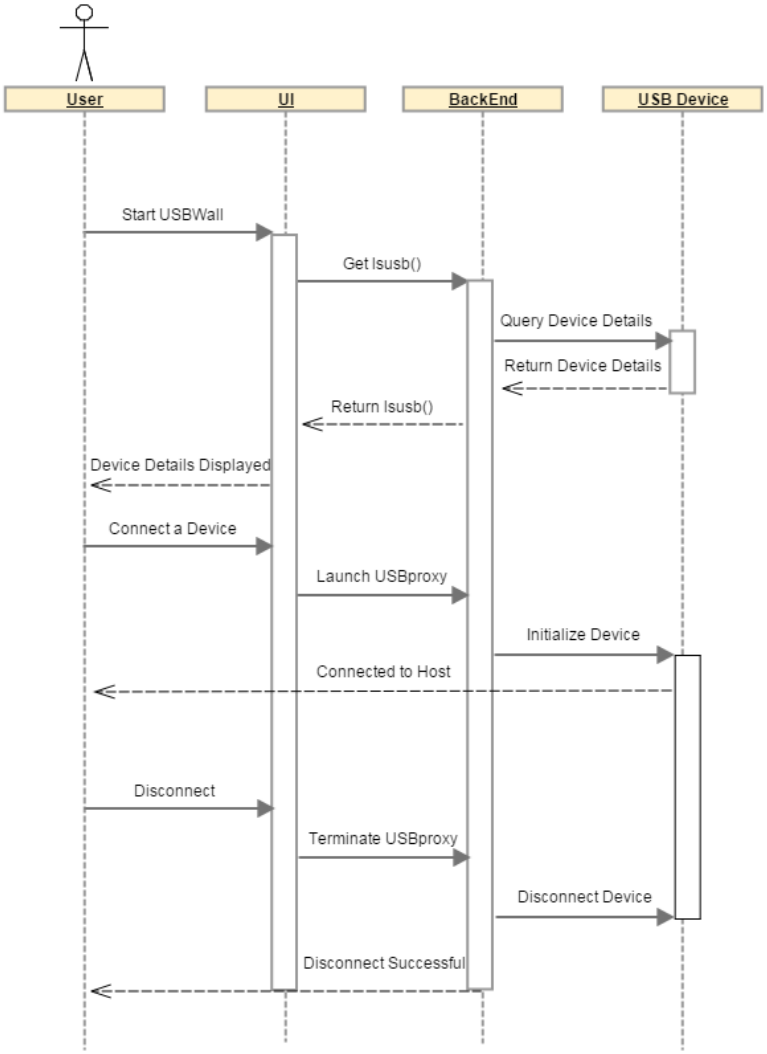


Figure 3.3: Sequence Diagram of USBWall

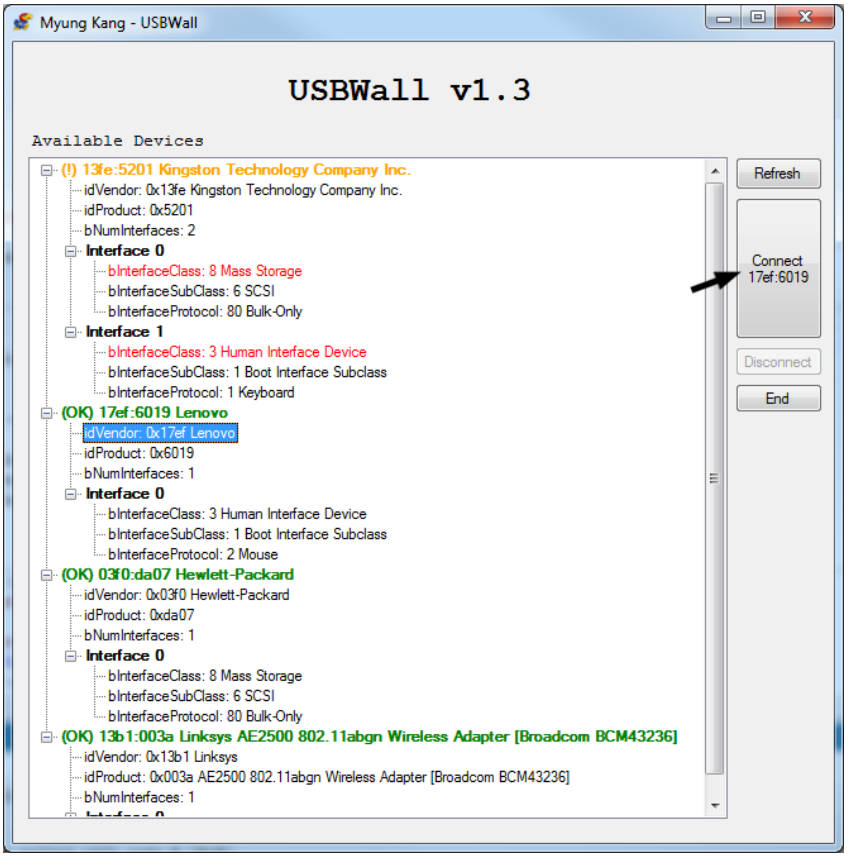


Figure 3.4: Middleware Front End (UI) on Windows host

Since 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 of keystrokes from the bogus device is not transmitted to the host. In other words, when the enumeration at BBB is completed, the device will try sending keystrokes to the host in which the device is plugged. In this case, BBB is the host. BBB's local console, to which the device's input is directed, remains at the linux's standard login screen. Therefore, any keystroke attempts sent to the host is directed to BBB instead, which ignores unless the keystroke precisely matches the login information of a user who has a 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' details. 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 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 [31], an open source project by Dominic Spill, is used to emulate the function of the device on BBB's host-facing interface using `gadgetfs` linux subsystem. USBProxy is launched with the PID and VID of the device to ensure only the selected device is relayed. To emulate the device functionalities, USBProxy utilizes `gadgetFS`, which is part of BBB's Debian 3.12.0-bone8 operating system.

3.2.3 Expected Protection

Using 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 BBB via SSH terminal, the UI obtains the information of the device plugged in before it is 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 3.5 shows the source code that issues the `lsusb -v` command to BBB to obtain USB devices' information from BBB to the host. The result is parsed and stored into a structure for later display.

```
Private Function lsusb() 'returns lsusb in struct in strings
    Dim templsusb As String = sshruncmd("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 bIntNumforFor 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
            bIntNumforFor = CInt(line.Substring(24))
            ReDim Preserve arr_lsusb(i).bInterface(bIntNumforFor)
        End If
        If line.Contains("bInterfaceClass") Then
            arr_lsusb(i).bInterface(bIntNumforFor).bInterfaceClass = line.Substring(31)
        End If
        If line.Contains("bInterfaceSubClass") Then
            arr_lsusb(i).bInterface(bIntNumforFor).bInterfaceSubClass = line.Substring(31)
        End If
        If line.Contains("bInterfaceProtocol") Then
            arr_lsusb(i).bInterface(bIntNumforFor).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 3.5: USBWall's Snippet of Issuing and Parsing `lsusb`

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 to launch 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.

3.3 Summary

This chapter discussed the identification and characteristics of USB devices with malicious firmware. Close analysis of USB enumeration process reveals that the vulnerability exists in low level transactions which are handled by the controllers of the host and the device. Absence of the universal access to the device's firmware storage adds the complexity to the vulnerability. Devices with malicious firmware can be identified when they appear as a different type of device than its physical characteristics, such as posing as a keyboard when it is a flash storage device.

Reacting to the discovery that this new attack can only be detected after USB enumeration occurs, we design USBWall to enumerate the device on behalf of the host. USBWall protects the host by delegating the USB enumeration process to BBB middleware. The suspected device is plugged into the BBB, then the enumeration details are stored to be sent to the host. Such sandboxed operation offers an effective separation between the host and the suspected device. USBWall itself is also protected from the attack as its local console, to which any malicious firmware attack execution is directed, waiting for the login details. Unless the input matches precisely with the username and the password of the local user, no action is executable at BBB. When USBWall UI receives the enumeration details from the BBB, it parses the information and color-codes certain entries to highlight the features of the device so that the user can decide easily. After the user reviews the information of the suspected device, user can choose to connect the device using USBWall UI. We expect the protection to be effective due to independent enumeration transactions, as well as the BBB's ability to protect itself from malicious devices.

Chapter 4

Validation for Sandboxed USB Enumeration

This chapter discusses the results and analyzes of the effectiveness of USBWall against USB devices with 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 successfully runs. The HID payload is designed to launch a series of keystrokes which will cause a Windows machine to run the notepad application, then type predefined characters [3]. 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 [19], we test 100 MB data transfer throughput for sequential, 512 KB, and 4 KB. We use the average value of five test results for better confidence.

4.1 Experiment Environment

USBWall is developed in Linux, and the Windows user interface portion is written in Visual Studio 2012 professional on Windows 7 64-bit version. Linux is used on 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 [15].

We compare USBWall's protection with AVG, avast!, Windows Defender, and with no antivirus installed for control. [4, 23, 32] The test firmware is written to launch a notepad using Rubber Ducky script [14]. If a notepad launches and keystrokes are entered without any user intervention, we consider the protection to be ineffective as it allowed the device to run a arbitrary keystrokes. Figure 4.1 shows the notepad launched on the host with no antivirus when a test device is plugged in. Upon plugging in, notepad is opened. Without any further user actions, keys are typed into the notepad. We use this scenario to determine if the each protection is effective against BadUSB attack.

File transfer throughput is also measured by performing benchmark tests with CrystalDiskMark (CDM) [19]. CDM performs read and write operations in a predefined size of data at different sizes of blocks. We run tests five times and average the result. To test the throughput when USBWall is in use, we first connect a USB

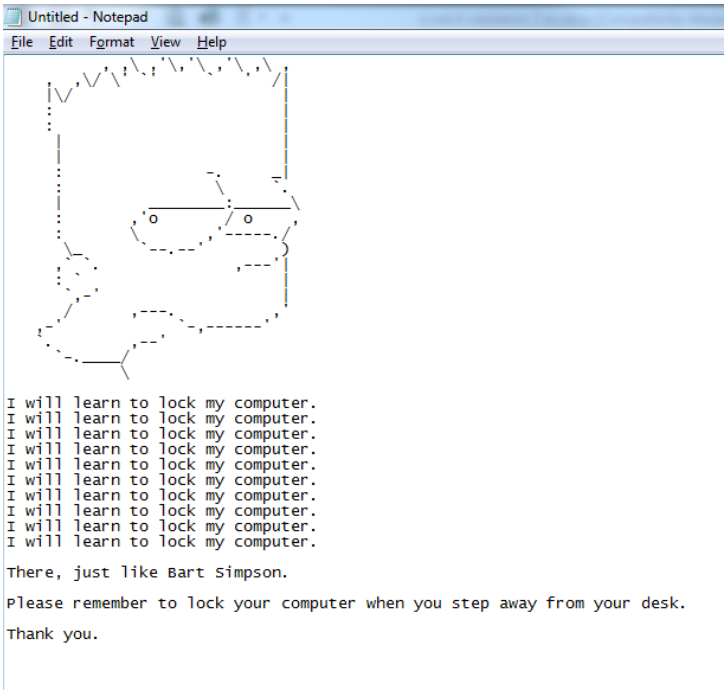


Figure 4.1: Screenshot of Successful Launch of HID Payload Launch

device to BBB. 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 BBB via `ps -ef` command on 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.

Type	Sub Type	Specifications
Host Computer	CPU	Intel®Core™2 Q9650 3.00GHz
	RAM	8 GB
	Storage	1 TB
	Operating System	Windows 7 Professional 64-bit
	USB Controller	Intel X48 Chipset with ICH9R
USBWall (BBB, Middleware)	CPU	Sitara XAM3359AZCZ 1GHz
	RAM	512MB
	Storage	2GB eMMC
	Operating System	Debian 3.12.0-bone8
	Power Supply	5V 2A with 5.5mm x 2.1mm Plug
Development Tool	User Interface	Visual Studio 2012 Professional
Sample USB Devices	Psychson-Applied Simulated BadUSB Drive	Toshiba TransMemory 16GB PFU016U1BCK
	BadAndroid-v0.1 Device	Samsung SPH-D700

Table 4.1: Specifications of Components Used in Testing

4.2.1 BadUSB Devices with Commercially Available Antivirus

To test and compare the effectiveness of different protections, a Psychson HID payload example [15] BadUSB device is presented to hosts with different antivirus protections which are commercially available 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 allowed the BadUSB's firmware to launch its preset keystrokes. The results show 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 protection installed. Expectedly, the firmware flawlessly runs on the host, opening a notepad and typing preset keystrokes. This result is compared to other test cases with antivirus softwares.

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 [32]. (Database version 4306/9296, AVG Antivirus FREE 2015.0.5751) Even with all real-time protection offered by the antivirus is enabled, BadUSB is able to run without any user interaction. Figure 4.2 shows the successful launch of the firmware's preset texts with AVG running.

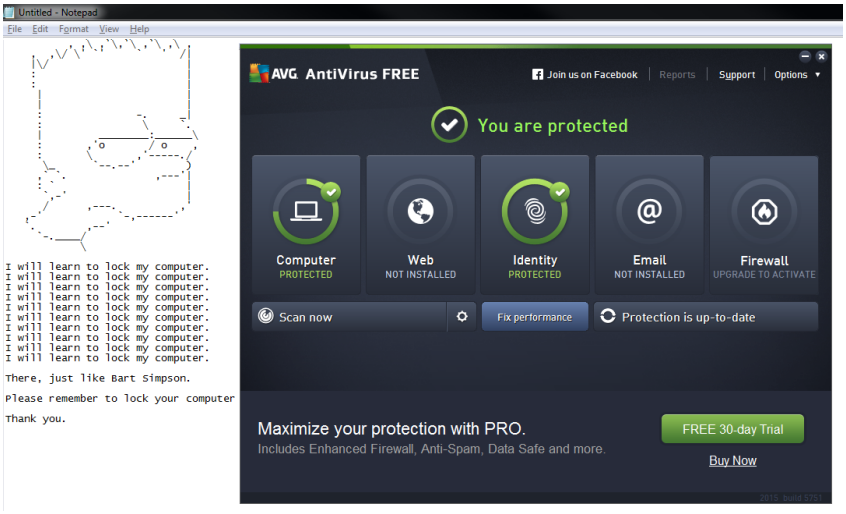


Figure 4.2: BadUSB HID Payload Launched with AVG Antivirus

Config 3: avast! by Avast

Next, the host is equipped with avast! antivirus software by Avast. The antivirus is then allowed to update to the most recent database pattern at the time of writing [4]. (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 4.3 shows the successful launch of the firmware's preset texts with avast! running.

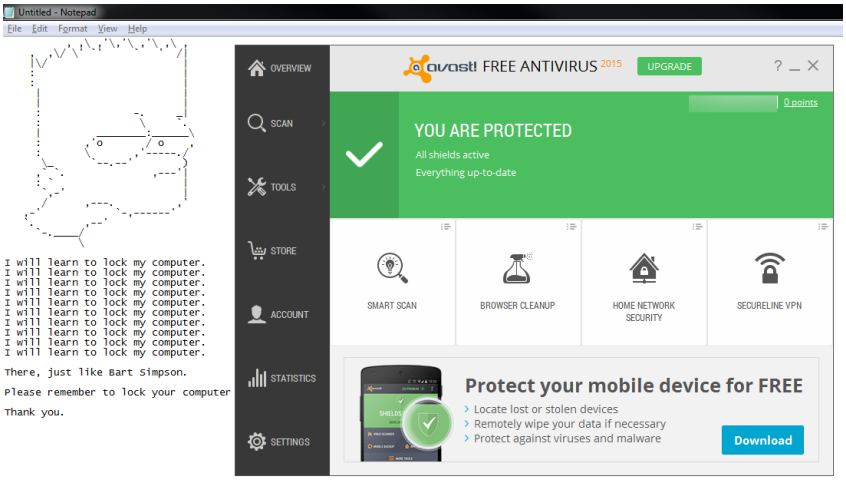


Figure 4.3: BadUSB HID Payload Launched with avast! Antivirus

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 was updated to the most recent database pattern at the time of writing. (Database 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 4.4 shows the successful launch of the firmware's preset texts with Windows Defender running.

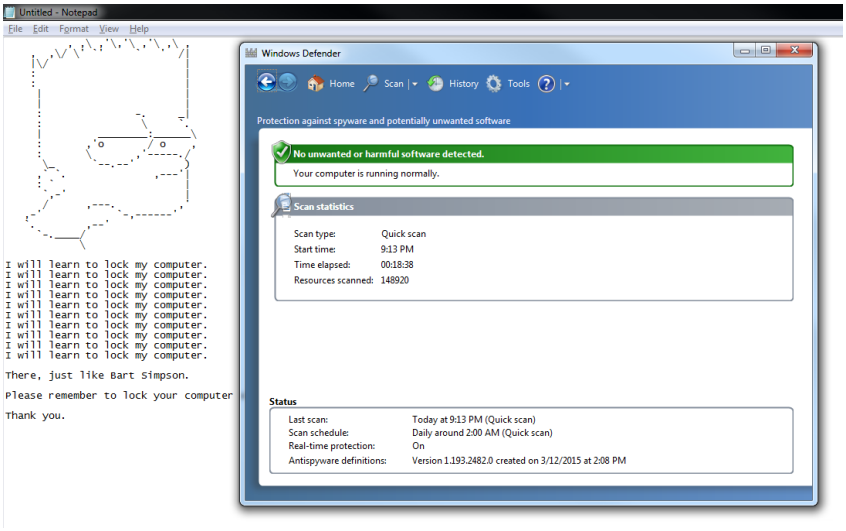


Figure 4.4: BadUSB HID Payload Launched with Windows Defender

4.2.2 BadUSB Devices with USBWall

The previous tests show that the current antivirus approach is ineffective in preventing the BadUSB attack. This is not only due to the nature of the attack that exploits the process during enumeration, but also to the access to the malicious firmware storage area that is proprietary. Lacking a standard way to access the firmware storage area, it remains largely impossible to antivirus 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 [15]. Secondly, 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.

Psychson Devices with USBWall

Toshiba TransMemory 16GB is a small flash drive which 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 a HID Keyboard. User launches USBWall by double-clicking the icon on the desktop after plugin of the suspected device. At startup of USBWall, it shows the tree list of connected devices as shown in Figure 4.5, obtained by BBB using `lsusb -v` command.

As shown in Figure 4.5, it is not only possible, but also easier, 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 of BadUSB attack. As the USB handshake only happened at the protected level of BBB, user can disconnect the device safely.

Although we assume all device as unsafe, we believe that user must have a way to override the warning and initiate the device. Therefore, user still has the 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.

BadAndroid Devices with USBWall

BadAndroid script made available by SRLabs turns an android phone into a network adapter for Man-in-the-Middle (MITM) attack [20]. The script is launched on Samsung SPH-D700 Android phone before connecting the microUSB cable to BBB. When BadAndroid-enabled mobile device is presented to a Windows 7 host with

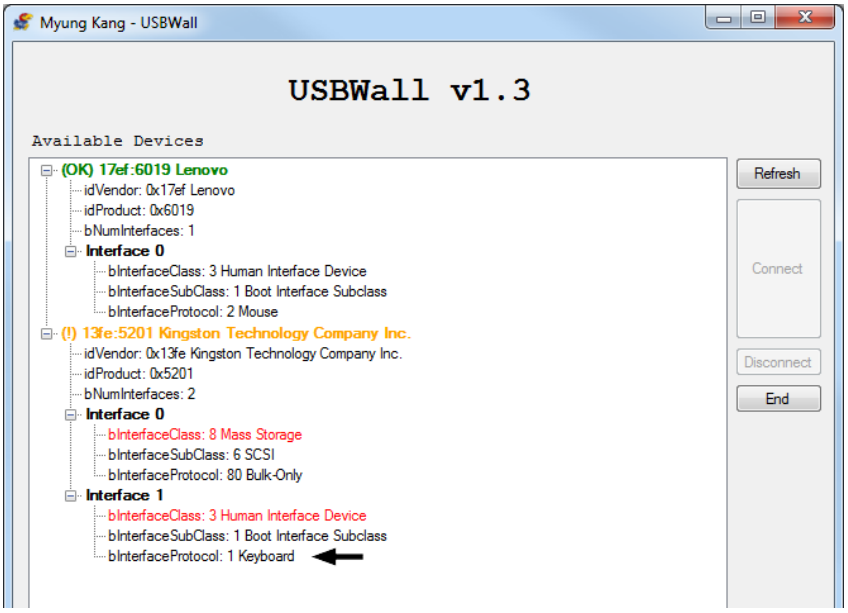


Figure 4.5: USBWall Detection of Psychson-Applied Sample BadUSB Device

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 startup of USBWall, it shows the tree list of connected devices as shown in Figure 4.6, obtained by 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 host if it is a valid device insertion. Figure 4.6 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 of the warnings provided by USBWall. User simply needs to click 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.

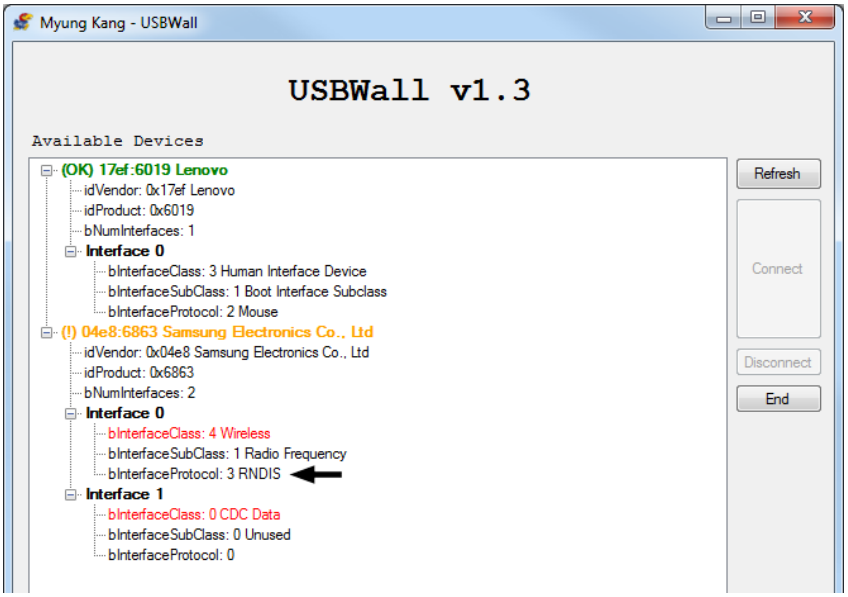


Figure 4.6: USBWall Detection of BadAndroid-Activated Mobile Phone

4.2.3 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 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+ 2GB 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 Table 4.2 and 4.3, 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, mainly due to the limited processor capability of BBB which is responsible for relaying the data. The copying

Type	Test Size	1	2	3	4	5	Average
via USBWall	Sequential	1.975	1.977	1.977	1.950	1.952	1.966
	512K	1.956	1.981	1.981	1.936	1.955	1.962
	4K	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
	512K	21.28	21.35	21.23	21.24	21.13	21.25
	4K	5.560	5.564	5.586	5.532	5.510	5.550

Table 4.2: CrystalDiskMark Result (Read, 100MB, MB/s)

Type	Test Size	1	2	3	4	5	Average
via USBWall	Sequential	1.890	1.886	1.889	1.907	1.906	1.896
	512K	1.159	1.145	1.173	1.147	1.088	1.142
	4K	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
	512K	2.140	2.122	1.947	2.034	2.093	2.067
	4K	0.024	0.023	0.023	0.024	0.023	0.023

Table 4.3: CrystalDiskMark Result (Write, 100MB, MB/s)

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 4.7 shows the CPU utilization on 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 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.

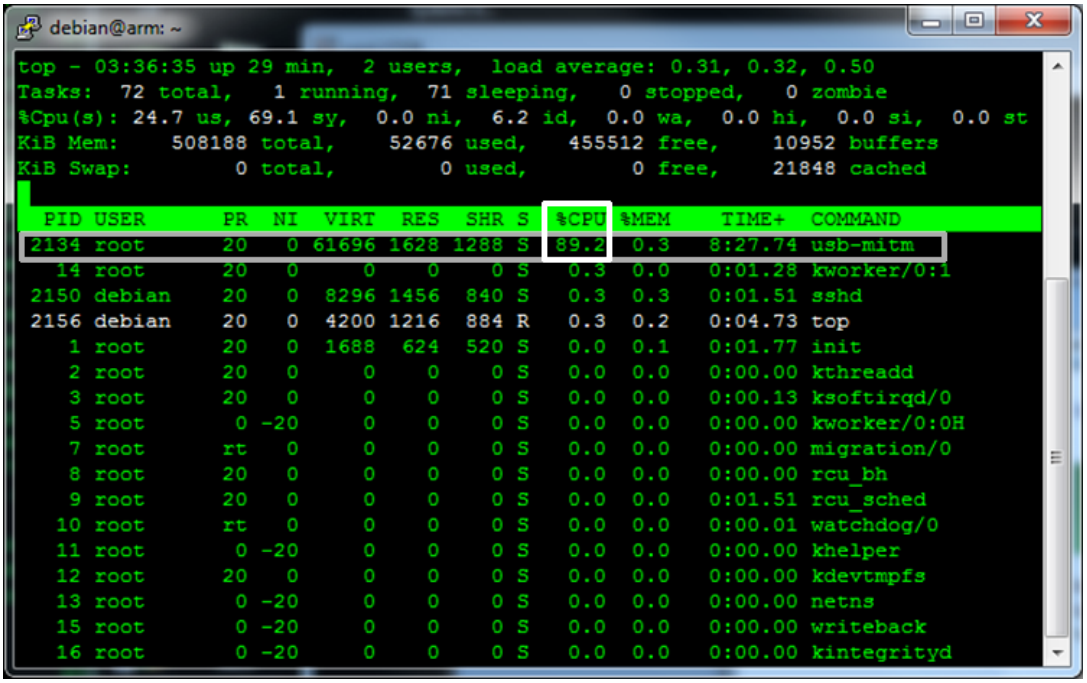


Figure 4.7: ps -ef of BBB While Transferring

4.3 Validation Conclusions

In this chapter, the implementation of USBWall is explained, and tested with Psychson and BadAndroid devices. AVG Free, avast! Free and Windows Defender are tested against the sample BadUSB devices. We showed that no existing antivirus protections are capable of blocking or detecting the presence of the malicious firmware. We assess that the difficulty of existing approach comes from the fact that the malicious firmware resides in the firmware storage, which are generally inaccessible to the user. Also, the access to firmware storage of a USB devices are mostly proprietary. Lacking the universal way to access the firmware storage area, the antivirus programs have no way to distinguish whether firmware's behavior is legitimate or malicious.

We show 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 only happened at the protected level of BBB, the user can disconnect the device safely without affecting the host's integrity.

While the conventional antivirus protections have 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.

	AVG Free 2015.0.5751	Avast Free Antivirus 2015.10.2.2214	Windows Defender 6.1.7600.16385	USBWall v1.3
Psychson-HID	No	**No**	No	**Yes**
BadAndroid	No	No	No	**Yes**

Table 4.4: Comparison of Protections against Sample BadUSB Devices

4.4 Hardware and Software Considerations

USBWall utilizes BeagleBone Black (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 the protection from any USB devices with malicious firmware, we find that the data transfer speed is reduced than compared to direct connection. Although the BBB has a 1GHz Siatara XAM3359AZCZ processor [8], a performance degradation is inevitable when it comes to the data throughput. This is because USBProxy [31] 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 upgraded BBB with faster hardware. Nevertheless, the integrity of transmitted data is not af-

fected. Therefore, we believe USBWall's effectiveness against BadUSB devices still remain 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 platforms, only the UI needs to be rewritten because the back-end communication and USB enumeration is done on BBB independently.

4.5 Summary

This chapter discussed the result of USBWall against the USB devices with malicious firmware. HID payload test sample and BadAndroid script are 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 are not visible by the operating system. USBWall, however, successfully prohibits the 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 used to bridge the connection and introduce the device to the host.

We also test the file transfer throughput to measure the impact of having an additional node on the USB data path. After analyzing the CPU utilization on BeagleBone Black (BBB), we find that the BBB's limited 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 other operating sys-

tems than Windows, we believe the porting effort is minimal because only the user interface needs to be ported.

Chapter 5

Contributions and Future Work

The system discussed in this thesis protects the computers from BadUSB devices by introducing a middleware, BBB, to relay the device enumeration information before host is exposed to the device. Although many creative methods are used in the current design, there exists 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 high level kernel entities to distinguish one device from one another. However, with BBB's access to the hardware-level information, it may be possible to fingerprint a device based on its electrical characteristics such as delays and power consumption patterns during the enumeration.

Secondly, USBProxy uses `gadgetFS`. After `lsusb` is displayed and the user decides to connect, USBProxy relays the data using gadgetFS subsystem. Although it is capable of viewing the properties of all devices, if a type is not supported by `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 BBB's overhead affecting performance in chapter 4.2.3. Not only

is the performance restricted to USB 2.0, but also to the capability of the CPU which decreases the performance further. As a newer BBB is released with faster hardware, we anticipate better performance.

Last, we strongly believe that the concept of introducing a middleware for a physical computer peripherals is a powerful candidate for offering protection against user devices. Especially with the emerging risks from hardware trojans [7] [6], the approach of USBWall can be extended to more interfaces than USB.

All scenarios above make an excellent candidate for works to be explored. By filling in the gaps of current restrictions, such as IEEE 1667 [13]. We firmly believe that it will contribute to safer computing from malicious user devices. We believe where IEEE 1667 has failed to reach, USBWall will reach to larger user base thanks to the minimal changes required on operating systems.

Appendix 1: A Picture of USBWall

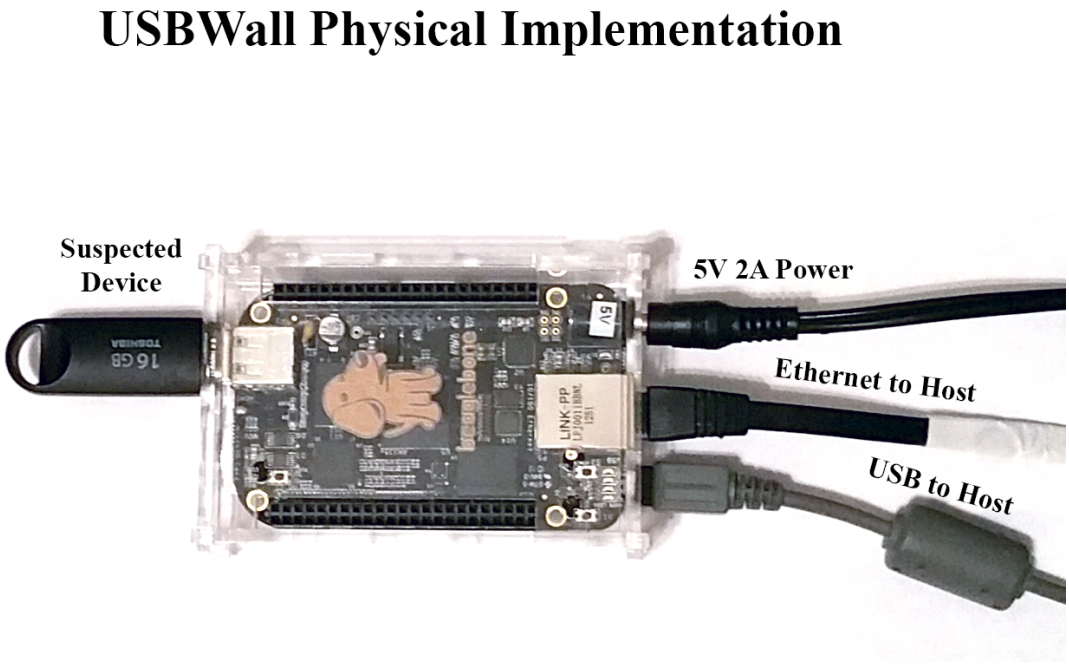


Figure 5.1: Picture of Physical Implementation of USBWall

Appendix 2: Rubber Ducky Script HID Example

```
REM Lock Me 0.3 - Script opens Notepad and types a message concerning locking the computer - by SurfKahuna (RJC)
GUI r
DELAY 200
STRING notepad.exe
ENTER
DELAY 300
STRING            , ,\ ,'\,'\ ,'\ ,\ ,
ENTER
STRING      ,   ,\/ \' ''      '    '   /|
ENTER
STRING      |\/                        |
ENTER
STRING      :                          |
ENTER
STRING      :                          |
ENTER
STRING       |                         |
ENTER
STRING       |                         |
ENTER
STRING       :                -.      _|
ENTER
STRING       :                 \       '.
ENTER
STRING       |          ________:______\
ENTER
STRING       :        ,'o       / o     ,
ENTER
STRING       :       \        ,'-----./
ENTER
STRING        \_      '--.--'          )
ENTER
STRING       ,'  '.               ,---'|
ENTER
STRING       : '                        |
```

```
ENTER
STRING        ‘,–’                    |
ENTER
STRING        /       ,–––.           ,’
ENTER
STRING     ,–’             ‘–,––––––’
ENTER
STRING     ‘.         ,––’
ENTER
STRING       ‘–.____/
ENTER
STRING              \
ENTER
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
STRING I will learn to lock my computer.
ENTER
ENTER
STRING There, just like Bart Simpson.
ENTER
ENTER
STRING Please remember to lock your computer when you step away from your desk.
ENTER
ENTER
STRING Thank you.
ENTER
DELAY 100
ALT SPACE
STRING x
```

Appendix 3: VB.NET Source Code of USBWall UI

```
Imports System.Text.RegularExpressions
Imports Renci.SshNet
Public Class Form1
    Private Structure str_lsusb
        Public EasyDesc As String
        Public idVendor As String
        Public idProduct As String
        Public bNumInterfaces As Integer
        Public bInterface() As str_bInterface 'redim when bNumInterfaces is known.
    End Structure
    Private Structure str_bInterface
        Public bInterfaceClass As String
        Public bInterfaceSubClass As String
        Public bInterfaceProtocol As String
    End Structure

    Dim sshhost As String = "192.168.123.110"
    Dim sshuser As String = "debian"
    Dim sshpass As String = "temppwd"
    Dim sshclient = New SshClient(sshhost, sshuser, sshpass)
    Dim shellstream As ShellStream

    Dim isUSBPrunning As Boolean = False

    Private Sub sshinit() 'set ssh env, start session
        Dim reply As String = String.Empty
        sshclient.connect()

        shellstream = sshclient.CreateShellStream("vt100", 80, 24, 640, 480, 1024)
        reply = shellstream.Expect("$", New TimeSpan(0, 0, 10))
        'wait for $ for 10 secs.
    End Sub
    Private Function sshruncmd(ByVal cmd As String) 'runs shell cmd and return result
```

```
        Dim reply As String = String.Empty
        Try
            Debug.Print("cmd: " & cmd)
            shellstream.WriteLine(cmd)
            reply = shellstream.Expect("$", New TimeSpan(0, 0, 5))
        Catch ex As Exception
            Debug.Print("cmd unsucessful: " & ex.Message.ToString)
            Return ex.Message
        End Try

        Return reply

    End Function
    Private Sub Button1_Click(sender As Object, e As EventArgs)
        'MsgBox(sshruncmd("ps -ef | grep usb-mitm"))
        test()
    End Sub
    Private Function lsusb() 'returns lsusb in struct in strings
        Dim templsusb As String = sshruncmd("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 bIntNumforFor 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
                bIntNumforFor = CInt(line.Substring(24))
                ReDim Preserve arr_lsusb(i).bInterface(bIntNumforFor)
            End If
            If line.Contains("bInterfaceClass") Then
                arr_lsusb(i).bInterface(bIntNumforFor).bInterfaceClass = line.
Substring(31)
            End If
            If line.Contains("bInterfaceSubClass") Then
                arr_lsusb(i).bInterface(bIntNumforFor).bInterfaceSubClass =
line.Substring(31)
            End If
            If line.Contains("bInterfaceProtocol") Then
                arr_lsusb(i).bInterface(bIntNumforFor).bInterfaceProtocol =
line.Substring(30).Trim()
            End If
        Next
```

```
81          ReDim Preserve arr_lsusb(i) 'when done, resize array before returning
            Return arr_lsusb
83      End Function
        Private Sub cmdLsusb_Click(sender As Object, e As EventArgs)
85
        End Sub
87      Private Sub lsusbToview()
            treeLsusb.BeginUpdate()
89
            treeLsusb.Nodes.Clear()
91
            Dim lsusbret As str_lsusb()
93          lsusbret = lsusb()
            Dim i As Integer = -1
95
            For Each device As str_lsusb In lsusbret
97              If device.EasyDesc.ToLower.Contains("hub") Then
                    Continue For
99              End If
                i = i + 1
101             Dim starintcount As String = "OK"
                For starwarningcount = 1 To device.bNumInterfaces - 1
103                 If starintcount = "OK" Then starintcount = ""
                    starintcount = starintcount & "!"
105             Next
                treeLsusb.Nodes.Add(New TreeNode("(" & starintcount & ") " & device
    .EasyDesc))
107             treeLsusb.Nodes(i).NodeFont = New Font(treeLsusb.Font, FontStyle.
    Bold)
                treeLsusb.Nodes(i).Nodes.Add("idVendor: " & device.idVendor)
109             treeLsusb.Nodes(i).Nodes.Add("idProduct: " & device.idProduct)
                treeLsusb.Nodes(i).Nodes.Add("bNumInterfaces: " & device.
    bNumInterfaces)
111
113             Select Case device.bNumInterfaces 'set color at device desc
                    Case Is > 3
115                     treeLsusb.Nodes(i).ForeColor = Color.Red
                    Case Is > 2
117                     treeLsusb.Nodes(i).ForeColor = Color.OrangeRed
                    Case Is > 1
119                     treeLsusb.Nodes(i).ForeColor = Color.Orange
                    Case Is = 1
121                     treeLsusb.Nodes(i).ForeColor = Color.Green
                End Select
123
125             For j = 0 To device.bNumInterfaces - 1
                    treeLsusb.Nodes(i).Nodes.Add(New TreeNode("Interface " & j))
127                 treeLsusb.Nodes(i).Nodes(3 + j).NodeFont = New Font(treeLsusb.
    Font, FontStyle.Bold)
                    treeLsusb.Nodes(i).Nodes(3 + j).Nodes.Add("bInterfaceClass: " &
     device.bInterface(j).bInterfaceClass)
129                 If (treeLsusb.Nodes.Item(i).ToString.Contains("OK") = False)
    Then treeLsusb.Nodes(i).Nodes(3 + j).Nodes.Item(0).ForeColor = Color.Red
```

```
131                 treeLsusb.Nodes(i).Nodes(3 + j).Nodes.Add("bInterfaceSubClass:
    " & device.bInterface(j).bInterfaceSubClass)
                    treeLsusb.Nodes(i).Nodes(3 + j).Nodes.Add("bInterfaceProtocol:
    " & device.bInterface(j).bInterfaceProtocol)
133             Next

135         Next
            'treeLsusb.ExpandAll()
137
            treeLsusb.EndUpdate()
139
    End Sub
141 Private Sub init()
        sshinit()
143     lsusbToview()
        pkillUSBP()
145
    End Sub
147
    Private Sub Form1_FormClosing(sender As Object, e As FormClosingEventArgs)
    Handles Me.FormClosing
149     cleanup()
    End Sub
151 Private Sub cleanup()
        pkillUSBP()
153     sshclient.disconnect()
        sshclient.dispose()
155


157 End Sub
    Private Sub pkillUSBP()
159     sshruncmd("echo temppwd | sudo -S pkill -f ""usb-mitm""")
    End Sub
161 Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.
    Load
        init()
163 End Sub


165
    Private Sub cmdConnect_Click(sender As Object, e As EventArgs) Handles
    cmdConnect.Click
167     Dim vid As String
        Dim pid As String
169     Dim anchor As Integer = 0

171     anchor = treeLsusb.SelectedNode.FullPath.IndexOf(":")
        vid = treeLsusb.SelectedNode.FullPath.Substring(anchor - 4, 4)
173     pid = treeLsusb.SelectedNode.FullPath.Substring(anchor + 1, 4)

175     runUSBP(vid, pid)
    End Sub
177 Private Sub runUSBP(ByVal vid As String, ByVal pid As String)
        sshruncmd("echo temppwd | sudo -S usb-mitm -v " & vid & " -p " & pid &
    " &")
```

```
179             isUSBPrunning = True
                cmdConnect.Text = "Connected with " & vid & ":" & pid
181             cmdConnect.Enabled = False
                cmdDisconnect.Enabled = True
183
            End Sub
185
            Private Sub cmdRefresh_Click(sender As Object, e As EventArgs) Handles
           cmdRefresh.Click
187             refreshview()

189         End Sub
            Private Sub refreshview()
191             lsusbToview()
            End Sub
193
            Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2
           .Click
195             Me.Close()

197         End Sub

199         Private Sub treeLsusb_AfterSelect(sender As Object, e As TreeViewEventArgs)
             Handles treeLsusb.AfterSelect
                Dim vid As String
201             Dim pid As String
                Dim anchor As Integer = 0
203
                anchor = treeLsusb.SelectedNode.FullPath.IndexOf(":")
205             vid = treeLsusb.SelectedNode.FullPath.Substring(anchor - 4, 4)
                pid = treeLsusb.SelectedNode.FullPath.Substring(anchor + 1, 4)
207
                showselecteddev(vid, pid)
209             enableconnectbtn()
            End Sub
211         Private Sub enableconnectbtn()
                If isUSBPrunning Then
213                 cmdConnect.Enabled = False
                    cmdDisconnect.Enabled = True
215             Else
                    cmdConnect.Enabled = True
217                 cmdDisconnect.Enabled = False
                End If
219
            End Sub
221         Private Sub showselecteddev(ByVal vid As String, ByVal pid As String)
                If isUSBPrunning Then Exit Sub
223             'update vid and pid for connect button
                cmdConnect.Text = "Connect " & vid & ":" & pid
225
            End Sub
227
            Private Sub cmdDisconnect_Click(sender As Object, e As EventArgs) Handles
           cmdDisconnect.Click
229             stopUSBP()
```

```
            cmdConnect.Text = "Select Device"
            cmdConnect.Enabled = False
            cmdDisconnect.Enabled = False
        End Sub
        Private Sub stopUSBP()
            pkillUSBP()
            isUSBPrunning = False
        End Sub
End Class
```

Appendix 4: Sample Result of lsusb -v on Test BadUSB Device

```
debian@arm:~$ lsusb -v

Bus 002 Device 002: ID 13fe:5201 Kingston Technology Company Inc.
Couldn't open device, some information will be missing
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x13fe Kingston Technology Company Inc.
  idProduct          0x5201
  bcdDevice            1.10
  iManufacturer           0
  iProduct                0
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           71
    bNumInterfaces          2
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0x80
      (Bus Powered)
    MaxPower              150mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           3
      bInterfaceClass         8 Mass Storage
```

```
      bInterfaceSubClass      6 SCSI
      bInterfaceProtocol     80 Bulk-Only
      iInterface              0
      Endpoint Descriptor:
        bLength                7
        bDescriptorType        5
        bEndpointAddress    0x81  EP 1 IN
        bmAttributes           2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0040  1x 64 bytes
        bInterval              0
      Endpoint Descriptor:
        bLength                7
        bDescriptorType        5
        bEndpointAddress    0x02  EP 2 OUT
        bmAttributes           2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0040  1x 64 bytes
        bInterval              0
      Endpoint Descriptor:
        bLength                7
        bDescriptorType        5
        bEndpointAddress    0x83  EP 3 IN
        bmAttributes           3
          Transfer Type            Interrupt
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0008  1x 8 bytes
        bInterval              0
    Interface Descriptor:
      bLength                  9
      bDescriptorType          4
      bInterfaceNumber         1
      bAlternateSetting        0
      bNumEndpoints            2
      bInterfaceClass          3 Human Interface Device
      bInterfaceSubClass       1 Boot Interface Subclass
      bInterfaceProtocol       1 Keyboard
      iInterface               0
        HID Device Descriptor:
          bLength                9
          bDescriptorType       33
          bcdHID              1.01
          bCountryCode           0 Not supported
          bNumDescriptors        1
          bDescriptorType       34 Report
          wDescriptorLength     63
         Report Descriptors:
           ** UNAVAILABLE **
      Endpoint Descriptor:
        bLength                  7
```

```
        bDescriptorType         5
93      bEndpointAddress     0x83  EP 3 IN
        bmAttributes            3
95        Transfer Type             Interrupt
          Synch Type              None
97        Usage Type              Data
        wMaxPacketSize     0x0008  1x 8 bytes
99      bInterval               1
      Endpoint Descriptor:
101     bLength                 7
        bDescriptorType         5
103     bEndpointAddress     0x04  EP 4 OUT
        bmAttributes            3
105       Transfer Type             Interrupt
          Synch Type              None
107       Usage Type              Data
        wMaxPacketSize     0x0008  1x 8 bytes
109     bInterval               1
```

Bibliography

[1] Appavoo, J. and Hui, K. and Soules, C. A. N. and Wisniewski, R. W. and Da Silva, D. M. and Krieger, O. and Auslander, M. A. and Edelsohn, D. J. and Gamsa, B. and Ganger, G. R. and McKenney, P. and Ostrowski, M. and Rosenburg, B. and Stumm, M. and Xenidis, J. Enabling Autonomic Behavior in Systems Software with Hot Swapping. *IBM Systems Journal*, 42(1):60–76, 2003.

[2] I. Arce. Bad Peripherals. *Security & Privacy, IEEE*, 3(1):70–73, 2005.

[3] SurfKahuna at hak5darren. *Payload lock your computer message*. https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payload---lock-your-computer-message [Accessed 9/25/2014].

[4] Avast. *Avast | The World's #1 Antivirus Software*. https://www.avast.com/index [Accessed 9/25/2014].

[5] R. Brandom. *USB Has a Huge Security Problem That Could Take Years to Fix*. http://www.theverge.com/2014/10/2/6896095/this-published-hack-could-be-the-beginning-of-the-end-for-usb [Accessed 11/18/2014].

[6] Clark, J. On Unintended USB Communication Channels. Master's thesis, Royal Military College of Canada, 2009.

[7] Clark, J. and Leblanc, S. and Knight, S. Risks Associated with USB Hardware Trojan Devices Used by Insiders. In *Systems Conference (SysCon), 2011 IEEE International*, pages 201–208, 2011.

[8] Coley, G. *BeagleBone Black System Reference Manual Rev A5.6*, 2013.

[9] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips. Universal Serial Bus specification v2.0. *USB Implementers Forum, Inc.*, April 2000.

[10] Falliere, N. and Murchu, L. and Chien, E. W32. Stuxnet Dossier. *White paper, Symantec Corp., Security Response*, 2011.

[11] Future Technology Devices International Limited (FTDI). Simplified Description of USB Device Enumeration. *FTDI Ltd. Technical Note*, 1.0:14, 2009.

[12] G-Data. How to Be Sicher from USB Attacks. `https://www.gdatasoftware.com/en-usb-keyboard-guard` [Accessed 9/24/2014].

[13] IEEE 1667 Working Group. IEEE Standard for Authentication in Host Attachments of Transient Storage Devices. *IEEE Std 1667-2009 (Revision of IEEE Std 1667-2006)*, pages 1–125, March 2010.

[14] J. Hall. *Duck Toolkit v.2*. `http://www.ducktoolkit.com/Home.jsp` [Accessed 9/25/2014].

[15] Harman, R. Controlling USB Flash Drive Controllers: Expose of Hidden Features. In *ShmooCon 2014*, 2014.

[16] Intel Corporation and Microsoft Corporation. Legacy Plug and Play Guidelines. *A Technical Reference for Legacy PCs and Peripherals for the Microsoft Windows Family of Operating Systems*, 1.0, May 1999.

[17] Jin, Y. *Trusted Integrated Circuits*. Ph.D. Dissertation, Yale University, 2012.

[18] Karabarbounis, L. and Neiman, B. The Global Decline of the Labor Share. Technical report, National Bureau of Economic Research, 2013.

[19] Kashiwano M. *Crystal Dew World*. http://crystalmark.info/?lang=en [Accessed 9/25/2014].

[20] Security Research Labs. *Turning USB peripherals into BadUSB*. https://srlabs.de/badusb/ [Accessed 9/24/2014].

[21] A. Mammit. *How Bad Is BadUSB? Security Experts Say There Is No Quick Fix*. http://www.techtimes.com/articles/17078/20141004/how-bad-is-badusb-security-experts-say-there-is-no-quick-fix.htm [Accessed 11/18/2014].

[22] Microsoft. *Introducing Enhanced Storage Access*. https://technet.microsoft.com/en-us/library/Dd560657(v=WS.10).aspx [Accessed 11/18/2014].

[23] Microsoft. *Microsoft Security Essentials - Microsoft Windows*. http://windows.microsoft.com/en-us/windows/security-essentials-download [Accessed 9/25/2014].

[24] Mitra, R. The Information Technology and Business Process Outsourcing Industry: Diversity and Challenges in Asia. *Asian Development Bank Economics Working Paper Series*, 365(365), 2013.

[25] Oshri, I. and Kotlarsky, J. and Willcocks, L. *The Handbook of Global Outsourcing and Offshoring 3rd Edition*. Palgrave Macmillan, 2015.

[26] PCI Security Standards Council, LLC. *"Payment Card Industry (PCI) Data Security Standard: Requirements and Security Assessment Procedures, Version 3.1"*. May, 2015.

[27] Renci. *SSH.NET Library*. https://sshnet.codeplex.com/ [Accessed 9/25/2014].

[28] Rich, D. Authentication in Transient Storage Device Attachments. *Computer*, 40(4):102–104, 2007.

[29] Rueter, C. *The Cybersecurity Dilemma*. Master's thesis, Duke University, 2011.

[30] L. Spector. *BadUSB: What You Can Do About Undetectable Malware on a Flash Drive*. http://www.pcworld.com/article/2840905/badusb-what-you-can-do-about-undetectable-malware-on-a-flash-drive.html [Accessed 11/18/2014].

[31] Spill, D. and Stasiak, A. An Open and Affordable USB Man in the Middle Device. *ShmooCon 2014*, 2014.

[32] AVG Technologies. *AVG Free Antivirus & Malware Protection*. http://free.avg.com/us-en/free-antivirus-download [Accessed 9/25/2014].

[33] Tetmeyer, A. and Saiedian, H. Security Threats and Mitigating Risk for USB Devices. *Technology and Society Magazine, IEEE*, 29(4):44–49, 2010.

[34] Usher, A. *Pod Slurping*. http://www.sharp-ideas.net/pod_slurping.php [Accessed 9/25/2014].

[35] Verma, S. and Singh, A. Data Theft Prevention & Endpoint Protection from Unauthorized USB Devices. In *Advanced Computing (ICoAC), 2012 Fourth International Conference on*, pages 1–4, 2012.

[36] Zhaohui, W. and Johnson, R. and Stavrou, A. Attestation & Authentication for USB Communications. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*, pages 43–44, 2012.