OS Independent and Hardware-Assisted Insider Threat Detection and Prevention Framework

Enes Erdin*, Hidayet Aksu*, Selcuk Uluagac*, Micheal Vai† and Kemal Akkaya‡

* Cyber-Physical Systems Security Lab, Florida International University, Miami, FL {eerdi001, haksu, suluagac}@fiu.edu

† MIT Lincoln Laboratory Lexington, MA mvai@ll.mit.edu

‡ Advanced Wireless and Security Lab, Florida International University, Miami, FL kakkaya@fiu.edu

Abstract—**Governmental and military institutions harbor critical infrastructure and highly confidential information. Although institutions are investing a lot for protecting their data and assets from possible outsider attacks, insiders are still a distrustful source for information leakage. As malicious software injection is one among many attacks, turning innocent employees into malicious attackers through social attacks is the most impactful one. Malicious insiders or uneducated employees are dangerous for organizations that they are already behind the perimeter protections that guard the digital assets; actually, they are trojans on their own. For an insider, the easiest possible way for creating a hole in security is using the popular and ubiquitous Universal Serial Bus (USB) devices due to its versatile and easy to use plug-and-play nature. USB type storage devices are the biggest threats for contaminating mission critical infrastructure with viruses, malware, and trojans. USB human interface devices are also dangerous as they may connect to a host with destructive hidden functionalities. In this paper, we propose a novel hardware-assisted insider threat detection and prevention framework for the USB case. Our novel framework is also OS independent. We implemented a proof-of-concept design on an FPGA board which is widely used in military settings supporting critical missions, and demonstrated the results considering different experiments. Based on the results of these experiments, we show that our framework can identify rapid-keyboard key-stroke attacks and can easily detect the functionality of the USB device plugged in. We present the resource consumption of our framework on the FPGA for its utilization on a host controller device. We show that the our hard-to-tamper framework introduces no overhead in USB communication in terms of user experience.**

I. Introduction

Military bases and other governmental institutions put a lot of effort in order to protect their confidential data from attackers. For that purpose, they either assign different level of permissions to the users or implement firewalls or log activities or create honeypots. However, an intentional or unintentional information leakage by an insider makes the biggest impact on cyber assets as such perimeter defenses are, most of the time, useless for an insider.

Information assurance is usually provided by logging the activities or limiting the abilities of the users via software-based solutions at the operating system level, e.g., file activity monitoring software. Although the software-based loggers are useful in many ways, like easiness of implementation or maintenance or interoperability, an attacker with enough motivation and skills can circumvent such defensive barriers. In fact, defensive software products have deficiencies in theory. First of all, they are implemented on a proprietary hardware. One can not verify full trust unless she knows the full design details of the hardware being used. Secondly, in a computing system the software and the operating system used are important such that all low level protocols are implemented in the operating system and are brought into use as libraries or kernel-level modules; those libraries can easily be modified by malware or viruses. Third, the configuration of the software tools is also important; for example, a misconfiguration of a single firewall will cause catastrophic results when attacked. Indeed, every company is vulnerable to being hacked [1].

When it comes to preventing the information leakage, Universal Serial Bus (USB) devices are handy devices for transferring a vast amount of information from one source to another. Due to past notorious incidents (e.g., Stuxnet [2]) and its destructive behavior, USB flash disk usage is still limited and regulated for the last 15 years in military bases [3]. However, the attacks originating from the USB devices other than mass storage devices are becoming more complex and unexpected and yet easier as the firmware of the USB devices are more complex. Indeed the firmware of a specially tailored USB device can easily be modified in order to mimic human interface devices such as keyboards and mice [4], [5]. Additionally, those devices can open a network or a printer interface on the host machine in order to attack to the host or steal information from it.

In this paper, we propose a novel hardware-based insider threat detection and prevention framework focusing on USB. USB is the most powerful interface on a computing platform contrary to its underestimated power of flexibility. Indeed, organizations are investing in systems for network security for preventing attacks from outsiders; however, USB usage (where ability of turning into a network interface is only one out of many capabilities of it) is under the mercy of its user with its innate vulnerable nature. So, USB is the weakest yet strongest interface for the use of an insider.

To address these issues, we implemented a novel detection and prevention framework on a ZedBoard [6] hosting a Zynq system-on-chip (SoC) from Xilinx [7]. The power of Zynq comes from not only holding a dual core ARM processor (PS), but also utilizing a rich logic environment (PL) which can be served either for the use of the processor or standalone use. Note that, FPGAs are widely used in military missions supporting a wide range of forces, Air Force, Navy, Army, etc. In our case, ARM processor is running a Linux desktop which resembles the PC under protection. Around the processor, in the programmable logic part (PL), we implemented a configurable and flexible USB sniffer and an independent monitoring unit observing the data going through the USB connection. As described previously, either the ARM processor or the Linux OS may be vulnerable to attacks. However, the logic core we designed monitors the USB communication and alerts the system administrator through an independent administrative channel so that reconnaissance can be done more efficiently.

Contributions: Our contributions are as follows:

- We implemented a novel USB communication sniffer as a detection and prevention framework on the PL part of Zynq SoC. The implementation can serve for high-level synthesis implementation well with small modifications.
- The designed hardware core is flexible for serving the needs related to the security policy of the organization.
- Since the offered design is fully transparent and isolated from the main system, it creates no performance related issues, no overhead in terms of consuming system resources.
- The design supports low speed, full speed, and high speed data transfers.
- Thanks to being just a physical addition to the system, it is independent of the OS of the host, and it can be further developed as a tamper proof defense mechanism.

Organization. The paper continues as follows: In Section II, the background for the USB is discussed. In section III the threat model is introduced. In Section IV, the test-bed is introduced. In Section V, some use cases and results for related experiments are presented. After summarizing related works in Section VI, the paper is concluded with Section VII.

II. Background

In this section we will briefly describe the USB communication protocol, its components and will define the term “insider threat” in addition to well-known precautions for preventing data leakage caused by insiders.

In the first years of modern computing simple peripheral ports such as serial ports and the parallel printing ports were enough to handle the data transfer between PC and the external devices. Due to the increasing data amount to be transferred and increasing capability of the peripheral devices with new functionalities, USB emerged. The introduction of USB was apparently one of the greatest advances in the communication technology. Recently, most of the devices are shipped with only USB and other I/O connections like video or network connectors are omitted.

USB brings an undeniably rich user experience with its built in functionalities. Power of being functional brings the complexity. For example, for the Linux kernel, functionality with flexibility caused a complex layered architecture for handling the USB connection. Figure 1 shows the visited layers and loaded drivers when handling a simple USB keyboard in GNU/Linux OS; when a USB device is connected to a computer, signal level change in the bus creates an interrupt at the Host Controller. After recognizing that change, the initial messaging starts. With suitable detection of the device, final required drivers are loaded and the user starts to use the device through the desired application. Being flexible comes with a vulnerable nature which makes USB a good source of attack. A smartly designed attack can open a serial port terminal over TeleTYpewriter (TTY) on victim’s PC with root access, or a USB can be programmed to mimic behavior of other devices in order to create fake inputs.

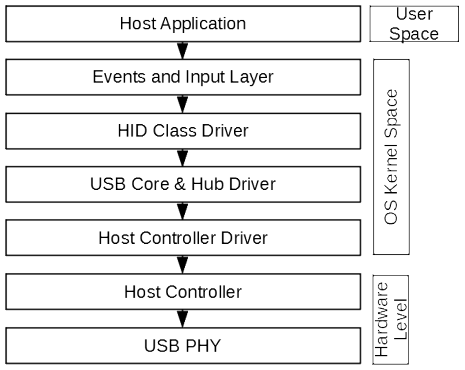


Fig. 1. An overview of layers visited and drivers loaded while handling a USB Human Interface Device (HID) in GNU/Linux OS.

USB is designed to operate in a master-slave fashion where a host machine initiates and controls the communication between itself and the USB device. With the introduction of the USB on-the-go standard, USB devices which are traditionally thought to be slaves are able to act as hosts and can interact with slave devices on some occasions, e.g., smart phones. USB 3.1 standard also started a new era in USB history with extra capabilities like charging laptops while connecting displays via it [8]. The new capabilities make USB standard hard to vanish in the near future.

USB Communication Protocol. USB interface is the de facto peripheral interface in computer systems. From network to human interface device communications to printing to camera applications, USB is serving users well as it was aimed. The advancement in both communication bandwidth, manufacturing capabilities and physical connectors with holding backward compatibility USB will be used for decades to come. Because of its popularity and ubiquity, we picked the USB connection as the main source for an insider threat since, if not secured, it is the most convenient way for getting lots of information from a computer.

ULPI interface. The first USB transciever designs were shipped with Macrocell Interface (UTMI) as a digital interface. Since UTMI was consuming a big area because of the pin count, lower pin count version of that intarface was introduced, UTMI+ Low Pin Interface (ULPI).

Host Controller Interface (HCI). Host controller is the physical interface for connecting a device (USB in our case) to a computer. It is basically the first level where the USB signaling is converted into register transfer level signals for the use of the operating system. HCI is the specification defining how the OS should interact with the host controller. For USB case, Open, Universal, Enhanced and Extensible host controller interfaces (OHCI, UHCI, EHCI, XHCI) are the four mostly utilized specifications. The target of those HCIs are different USB specification versions. We believe that realization of our proposed design is best suited for the usage in host controllers.

USB Device Enumeration. When a device is first plugged into a computer, change of voltage levels creates an interrupt assertion in the host controller. The controller identifies the specification version of the device. When the device specification version is identified the initial message exchange starts. In USB communication, the slave can not do anything until the master asks for it to do.

- *First message blocks:* The host asks for the "Device Descriptor" from the attached device. The address of the device is initially 0 by default. The device descriptor is the first crucial step for identifying the device type.
- *Second message blocks:* Host identifies the number of interfaces the device has. Additionally, the host assigns an address to the attached device. The interfaces can be thought as the definition of capabilities that the attached USB device has. For example, a simple web-cam device has camera and microphone capabilities, those require different interface drivers. With these messages the host learns about the device by requesting configuration descriptors, interface descriptors and endpoint descriptors as needed.
- *Remaining message blocks:* After getting all the capability information from the device, the OS loads the required drivers and applications start using the capabilities of the device in the way it is defined in the device driver.

For more information, we advise the readers to visit the USB Implementers Forum (USB-IF) [9].

USB Packets. There are four different kinds of USB packets: token, data, handshake, and start/end-of-frame packets and four kinds of transfer types: isochronous, interrupt, control, and bulk packet transfers. Each packet is identified by observing the 8-bit packet identification (PID) field in the beginning of corresponding packet. Packet transmission starts right after the transmission of synchronization bits. Synchronization bits are utilized for the use at the electrical level in order to align the data sampling clocks. So in practice, PID is the first byte carrying information in a given USB packet.

III. Threat Model

This section defines the insider threat problem and describes the threat model. The term "insider threat" is used to define the potential of data breach or harm to the assets of an organization due to the actions carried out by an employee. The intent of the employee can either be malicious such that the she wants to give harm to the organization or she may be innocent that she may harm the company accidentally. Insiders with malicious behavior can steal the confidential data of an organization or can create a backdoor for the outsiders for a more systematic and persistent attack.

We consider an adversary (i.e., insider) who has restricted access to a computing system. The adversary initiates physical attacks to the host such as plugging a completely forbidden or previously allowed but later modified USB device. The attacker has physical access to the system; however, entrance to the facility is controlled; hence, she can alter the device hardware or firmware but, she can not completely replace sub-systems with new ones. This type of adversary might exist in facilities where the computers' USB ports are directly reachable. Additionally, computers can be protected by BIOS passwords and the computer can be initialized by a live operating system after taking required steps to change the BIOS settings. Network connections to the computers are allowed which creates another attack surface for the hosts. The ultimate target of the attacker can be either exfiltrate data from the host device or suspend the system.

Attacker can take the following specific actions in order to be successful:

- **Modified device insertion.** The adversary can plug in a previously allowed but later modified USB device. BadUSB or USB devices with modified firmware fall into this type of attack.
- **Unauthorized Devices.** This type of devices can be used after eluding the software related security barriers. An example of this type of attack can be starting the host with external boot option so that the attacker can run a live OS in the victim host.
- **Access manipulation.** The attacker can modify the firmware of the USB devices such that she can alter the capabilities of the device. USB device functionalities other than usual human interface device or mass storage device fall into that type of attack. Among those are added microphone, headphone or network functions to the device. Super-human typing, e.g., rapid key strokes and rapid scrolling also fall into this category.

IV. Design Overview of the Framework

In this section, we detail the technical background of our framework. In the heart of our design, there is a USB data sniffer implemented in the logic part of the FPGA. Ideal way of sniffing the data packets on the bus is transforming them into lower speed digital signals like UTMI or ULPI packages. The USB data are turned into ULPI signals. Based on the level change in the control signals, the data capture between host and the device is initiated. The USB enumeration process is observed by utilizing finite state machines accordingly. Any kind of descriptor supplied by the USB device is stored. The rules supplied by the system administrator are also stored in the logic part. The communication interface between the administrator and the system can be any of the current communication protocols. For the sake of the security it is advised that communication should be other than that used in the entire organization for networking. In our case just to show the proof-of-concept, we used switches and the LEDs on the board. The rules are input by the administrator in order to allow or block certain type of devices. These rules can be

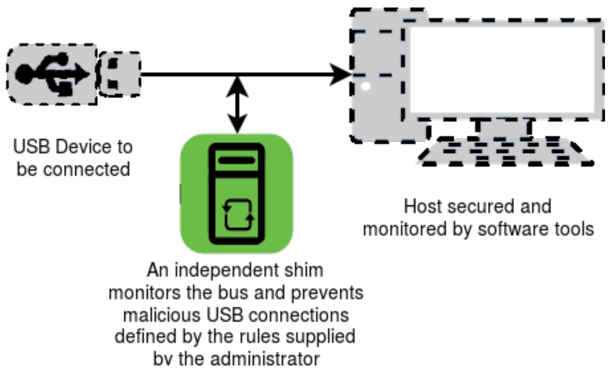

USB Device to be connected
Host secured and monitored by software tools
An independent shim monitors the bus and prevents malicious USB connections defined by the rules supplied bv the administrator

Fig. 2. Overview of the proposed framework.

manufacturer and the product IDs or they can be specific interface descriptors like keyboards, mice, printers, mass storage devices and so on. When the rule matches with a property of a connected USB device the design detects it and disrupts the communication, namely, it resets the communication. The best place for utilizing such a design is the host controller, where every connection reaches out there before being served by the operating system. This proof-of-concept design shows applicability of our approach.

The proposed framework can be seen in Figure 2. In the figure, the PC is the computer which is protected by the organizational security policy, USB device is the device that will be attached to the PC, and the shim icon at the lower side represents the control unit which resembles the system security administrators in practice. Defining the communication interface with the framework is not covered in this study. In our case, monitoring is done via Logic Analyzer and control is carried out via switches and buttons on the ZedBoard.

We used ZedBoard from Digilent [6] as the testbed. ZedBoard's Zynq has a hardly implemented ARM processor (PS) core with a high speed USB interface attached to it. That built-in USB interface of the Zynq SoC had given us a great idea for implementing our design on the ZedBoard. However, the USB interface of the processor can not be reached from the logic part (PL) of the FPGA due to the limitations brought by multiplexed input/output (MIO) and extended MIO (EMIO) configuration options, by default design of the SoC. As a solution to that problem, an external USB physical layer USB3300 PHY is utilized [10]. USB3300 is a USB2.0 compliant USB PHY which is widely used in commercial products.

VHDL is used for programming the FPGA and verification is carried out by tests describing various scenarios directly on the hardware via Vivado integrated logic analyzer (ILA).

In order to make the implementation hassle-free some points should be clarified related to hardware design issues.

- Signal Integrity (SI) Issues. For a successful communication between the ZedBoard and the USB3300 evaluation board, one should be careful about the SI related issues. Since the USB PHY interface is using a clock of 60 MHz frequency, in order to mitigate the SI related problems we designed a custom board targeting the most possible shortest signal trace. Long connections done with jumper cables will be problematic. On the other hand, ZedBoard is shipped with a serial 200 Ω resistors placed on the general purpose test connectors. In order not to cause a decrease in the signal strength, these resistors are replaced by 0 Ω resistors.
- ZedBoard configuration. The processor part of the Zynq is configured with the ability of running a full featured Linux Desktop. The Linux kernel used is supplied by Xilinx. The root file system is picked to be the Linaro desktop (a debian based Linux distribution) which is known to be supporting ARM devices [11]. So the final design demonstration includes a Linux desktop with a valid USB interface on the processor part and an active USB packet sniffer on the logic part controlled and monitored via switches and LEDs.

In the PHY part, observing a level change in "data direction" pin, PID data can be captured. Captured PID type triggers a finite state machine which gives data packets to the output of the module. Those outputs are fed in to another finite state machine so that the enumeration process is observed and validated successfully. After observing the packets system can infer the addresses of the USB devices, vendor IDs, device IDs, number of endpoints, type of endpoints etc. If administrator wants to prevent a particular type of device or endpoint, she can define rules through the configuration input.

V. Performance Evaluation

In this section, experiments related to different use cases and their results are discussed. During the evaluation we try to find answers to the following questions: (1) whether if we can measure the packet inter-arrival times in order to detect fast key stroke-based attacks, (2) if we can identify the USB device based on advertised descriptors, (3) what the resource consumption of our framework on the FPGA will be, (4) how we can hinder the communication when a malicious behavior is noticed, (5) how we can verify our design, (6) how we can come up with a testbed, and (7) how the framework is not affected by the OS of the host.

A. Testbed Design

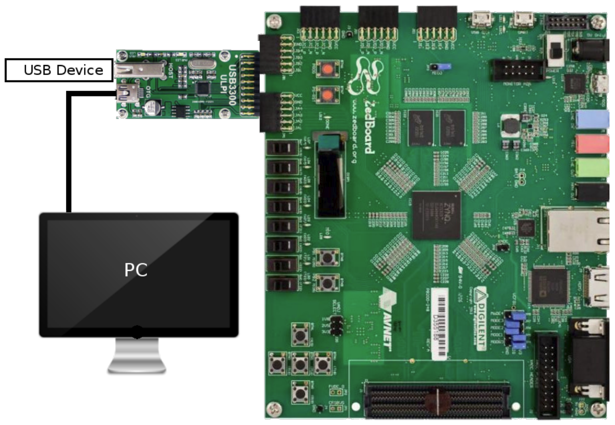
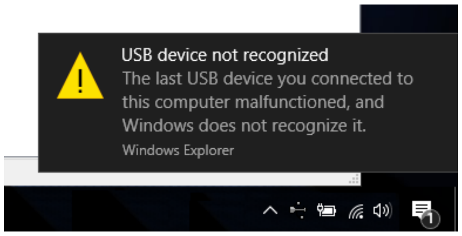


Fig. 3. Testbed setup to evaluate the framework. USB3300 is connected to the Zynq board via a custom interconnect board.

The final implementation of the proposed design is realized on a combination of ZedBoard and the USB3300 evaluation board as shown in Figure 3. The USB device is to be connected to the USB3300 PHY. The USB port of the computer is also connected to the USB3300 through a secondary connector. PHY is configured by the logic part of the SoC and it is assigned to be in the bypass mode.

B. Communication Disruption

As stated earlier in order to implement a fully functional detector, the design should be located in the host controller. Although our design can not be thought as a USB bridge, it can still prevent USB from connection by disrupting the USB bus. As USB specification points out, the USB devices can be put into reset mode by asserting $SE0$ signal state on the bus. This can be achieved by writing $0x50$ to the Function Control register of the PHY. Figure 4 is a screen capture after the USB bus is disrupted on a PC running Windows 10.



USB device not recognized

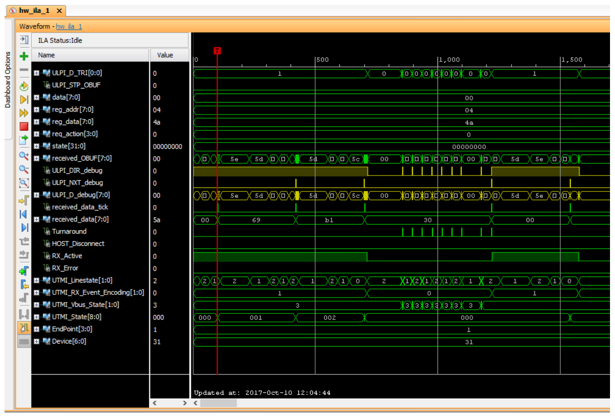
The last USB device you connected to this computer malfunctioned, and Windows does not recognize it.

Windows Explorer

Fig. 4. Unrecognized USB device warning after bus poisoning

C. Design Verification

Design is verified by conducting hardware level simulations via integrated logic analyzer (ILA) as seen in Figure 5 in conjunction with the *usbmon* dynamic kernel module [12]. Verification of the state machine based packet processing is done by the help of GNU/Octave scripting language [13]. The scripts are written in a way to simulate the hardware structure for easiness and consistency of verification.



hw_ila_1

Waveform - hw_ila_1

ILA Status:Idle

Name	Value
ULPI_D_TRI[0:0]	0
ULPI_STP_OBUF	0
data[7:0]	00
reg_addr[7:0]	04
reg_data[7:0]	4a
req_action[3:0]	0
state[31:0]	00000000
received_OBUF[7:0]	00
ULPI_DIR_debug	0
ULPI_NXT_debug	0
ULPI_D_debug[7:0]	00
received_data_tick	0
received_data[7:0]	5a
Turnaround	0
HOST_Disconnect	0
RX_Active	0
RX_Error	0
UTMI_Linestate[1:0]	2
UTMI_RX_Event_Encoding[1:0]	0
UTMI_Vbus_State[1:0]	3
UTMI_State[8:0]	000
EndPoint[3:0]	1
Device[6:0]	31

Updated at: 2017-Oct-10 12:04:44

Fig. 5. Integrated Logic analyzer output waveform during design verification

D. Rapid keyboard strokes

This use case is a realization of the peensy attack [5]. In this attack, the attacker loads a programmable USB device with a malicious file and rapid keyboard entries via creating a keyboard interface. This combination makes the attack highly destructive. The attack can occur either on insider's own PC or the attacker can infect an unattended PC quickly.

To realize this experiment, we programmed the Teensy board to mimic a keyboard in such a way that it sends malicious commands to the PC. The script helps Teensy open a terminal on Linux, modifies .bashrc file and closes the terminal. Steps are shown below:

- Send a $Ctrl + Alt + T$ combination to the PC. In GNU/Linux OS this opens a terminal window.
- Send $echo\ ''alias\ lsclone = ls''\ >> \sim/.bashrc''$ command as a payload which creates an alias named $lsclone$, basically a command which mimics ls command.
- Send $ALT + F4$ or $Ctrl + D$ key combination so that the terminal is closed.

In order to create a defense for such rapid keyboard stroke attack, we create a histogram of the inter-arrival times of the key strokes. If the frequency or number of the rapid keystrokes exceeds a previously defined threshold an alert is created. The rationale behind this defense is that USB HID devices are transmitted as interrupt data packets, and maximum frequency for interrupt packages is 1Khz in modern operating systems. A key is composed of two packets, one for hit the key and the other is relase the key. So in theory, one can send 500 characters per second with a programmed keyboard.

This simple attack takes around 600ms of which is the waiting time for terminal to be opened. The idea behind the experiment is demonstrating that an employee with a malicious intent can insert some malicious code to the computer of her colleague by only plugging a USB device in a very short time. During the experiments, it is realized that Teensy introduced itself as a keyboard with additional 2 interfaces where, an ordinary USB keyboard introduces only one interface which defines itself as a HID-Keyboard device. The same experiment replicated with the Rubber Ducky. Rubber Ducky successfully introduced itself as a keyboard device. That makes Rubber Ducky a tough target for defense.

E. Network interface

In this experiment, we try to identify different types of devices and interfaces. Since it creates a very large attack surfaces we picked USB network interface which is to be identified. We plugged in a network USB adapter in order to observe the enumeration process. The device introduced itself with interface class and subclass codes as FF which corresponds to a vendor specific class definition. We observed the device ID and vendor ID, it was a device manufactured by AX88179 Gigabit ehternet device from ASIX Electronics Corporation. For preventing (or allowing) that type of device, we created a filter to identify that vendor and device ID. Hence, the USB device with that vendor ID is prevented when plugged in.

F. OS independency

Due to the nature of its open structure, the solutions in the literature are based on modification of the Linux kernel. However, for insider threat detection, identification and prevention, we should keep in mind that around 50% of the companies are running in Windows OS [14]. Thanks to well defined USB and USB driver specifications, the design we propose is OS independent. We conducted all of our experiments in not only Linux, but also in Windows OS, too.

G. Evaluation and Results

1) Effects on data bit rate: Since our design is purely based on hardware flow, it does not have a direct effect on communication bit rate. In order to observe the overhead of the framework on effective bit rate, we transferred files from a host to a USB mass storage device. Figure 6 shows the results of file transfers with 3 different file sizes. In the experiments files of sizes 600 KB, 6 MB, and 256 MB are transferred. The mean of the duration of the transfers are plotted in Figure 6. From the figure, we see that our design does not introduce a significant latency in packet transmission.

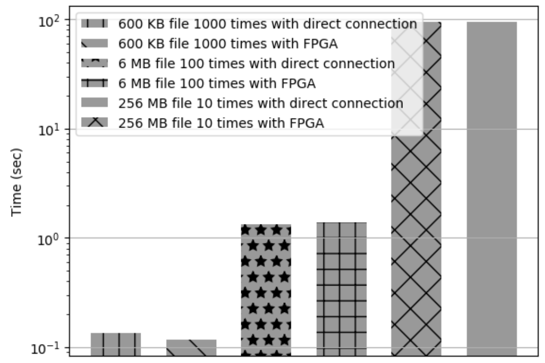


Fig. 6. Average time for sending different size files from PC to USB Flash disk with FPGA and without FPGA

2) Resource Allocation on the FPGA: In this section, the resource allocation of the FPGA design are discussed. The resources consumed in the Zynq SoC is shown in Table I. Achievable maximum clock frequency is out of discussion since the USB PHY we use has a fix 60 MHz clock. As a proof-of-concept limited number rules are defined, extending the rule table will introduce a slight increase in the resource allocation. For an increase in the number of rules, a block RAM utilization will be feasible. Resource utilization result offers us that extending a USB host controller with such capabilities is highly feasible and yet, it will be beneficial.

TABLE I
RESOURCE USAGE IN THE FPGA

Resource Name	Usage (count)
Slice LUTs	785
LUT as logic	497
LUT as Memory	288
Slice Registers - Flip Flops	390

VI. Related Work

In this section, we summarize the current literature about the studies conducted to attack hosts through USB ports and corresponding measures taken to prevent them. In order to prevent insider threats, the companies are taking extreme precautions. Some of these precautions are employee screening, legal obligations, education, duty separation, authentication, activity monitoring and so on. Due to the competitive nature of companies, much of those precautions are not satisfied, and insider threats still remain to be one of the biggest problems of the cyber world. For preventing data breach, both parties, the employee and the organization, have responsibilities. In that manner, the organization can make a stride in order to harden their defense line. We mention some of these seemingly efficient solutions.

- Activity monitoring and role-based authentication seems to be a fast and easy solution for keeping data secure as it relies on software based solutions. However, employee still has the potential to pacify that software based security burden and endanger the security of data and assets.
- Full disk encryption (FDE) is another solution for securing data. However, one should recall that the issue is the insider. Full disk encryption is only effective when the PC is initiated. After satisfying the login credentials the information on the PC is vulnerable to all kinds of attacks by the insider.
- Trusted platform module (TPM) is another instrument helping to satisfy the integrity of the sub-systems. Nonetheless as pointed out in FDE, TPM provides a benefit for keeping the integrity of the assets. Besides, successful attacks on TPM and FDE is another topic and it is not the concern of this paper.

The security of USB ports always has been a topic of discussion. Network attacks are prevented by complex systems built by service providers, firewalls, demilitarized zone implementations or built-in defence mechanisms by operating systems. However, due to its host level handling complexity, USB is a major source for attack, and the victim can be affected even by her own will by plugging a USB she found around [15]. USB mass storage devices still provide a good environment for spreading digital viruses to victims. For instance, Stuxnet is one of the most notorious and stealthy malware attack believed to be initiated by a USB storage device [2]. Different from the existing work, in our project we aim to filter the USB data connection through a harder-to-compromise independent channel by demonstrating its implementation on a hardware test setup.

In 2016, an employee was accused for stealing 50TB of data from National Security Agency (NSA) [16]. Tha attacker is believed to be using a sophisticated software that leaves no forensic footprint behind. He is reported that the insider might be using a USB-bootable operating system in conjunction with Tor or a VPN. In a similar attack an employee stole confidential data from his organization [17]. Aside from the fact that the insider took advantage of being a system administrator as an insider, he carried the information via a USB flash disk and made internal secrets public. BadUSB was the first practical demonstration of how evil a USB device can turn into by modifying its firmware [18]. In that study, it was shown that USB firmware can be re-programmed so that the device can behave in many different ways. For example, it behaves as a mass storage, which is expected by the user, but behind the scenes, it creates a network interface and inserts malicious commands to the host. Yet, with a device like Rubber Ducky USB, the hackers can create different attack vectors easily.

One similar application is the $PoisonTap$ Attack [19]. In [19], a RaspBerry Pi Zero mini single board computer was crafted for mimicking a USB device which maliciously siphons the web cookies, exposes an internal router and creates a backdoor for the attacker. The attack in the experiment is a persistent one in which even after the removal of the device the backdoor continues as intended. The attack detailed in [20] shows that by plugging a visually 'safe' USB flash disk into a Windows OS based PC an attacker can acquire the login and e-mail credentials of the PC. Indeed, the method is not new. Hacking passwords of many different applications by numerous attacks can be achieved via USB drives [21]. For preventing USB-based attacks authors in [22] suggests implementing a filter similar to the netfilter framework in Linux kernel which regulates the USB packet traffic. As a proof-of-concept Linux kernel of version 3.13 is modified. The framework gives user the ability to perform USB actions selectively. Although the idea is good as mentioned previously, it relies on pure software usage. In [23], the author implements a hardware-based firewall. The approach is using a two USB capable microprocessor and making them communicate via SPI ports. The design is made in order to mitigate the malicious behavior of the BadUSB network attack. The drawback of the design is it supports low speed and full speed communication.

VII. Conclusion

In this paper, we detailed, to the best of our knowledge, the first proof-of-concept design of a hardware-assisted insider threat detection and prevention module concentrated on USB on an FPGA and presented the results of conducted experiments for different use cases, namely, rapid keyboard stroke attack and USB device type detection. We also demonstrated the resource usage of the framework in the FPGA. The framework supports low, full and high speed communication specifications which is the limit of the PHY used throughout the experiments. However, the design can easily be extended to support recent USB 3.1 specification. We offer that the administration access should be done through an independent channel and our design is a realization of this idea. The design is fully transparent to the normal user and there is no performance limitations. In addition to all of these advantages, the design is free from the architecture and type of the operating system due to the nature of the USB communication standard. Since the design has full access to the USB PHY, custom USB packets can be created or monitored. That makes the ZedBoard design a good test bed for future USB studies. As a future work, packet detection based on behavioral analysis for an extensive identification will be studied.

Acknowledgment

This work was supported in part by NSF-CAREER-CNS-1453647 and NSF-1663051. The views expressed are those of the authors only.

References

[1] TechCrunch. (2017) Google's Heather Adkins thinks everybody is going to get hacked and you need to be ready. [Online]. Available: https://techcrunch.com/2017/09/18/googles-heather-adkins-thinks-everybody-is-going-to-get-hacked-and-you-need-to-be-ready/

[2] Kim Zetter. (2014) An Unprecedented Look At Stuxnet, The World's First Digital Weapon. [Online]. Available: http://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/

[3] United States Naval Academy. (2018, April) USNA Laptop, USB Flash Drive and External Hard Drive User Policy. [Online]. Available: https://www.usna.edu/PAO/documents/CSI

[4] HAK5. (2018, April) USB Rubber Ducky. [Online]. Available: https://hakshop.com/products/usb-rubber-ducky-deluxe

[5] Offensive Security. (2012, April) Advanced Teensy Penetration Testing Payloads. [Online]. Available: https://www.offensive-security.com/offsec/advanced-teensy-penetration-testing-payloads/

[6] Avnet. (2017) ZedBoard Zynq-7000 Development Board. [Online]. Available: http://zedboard.org/product/zedboard

[7] Xilinx. (2017) Zynq-7000 All Programmable SoC Product. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[8] Toms Hardware. (2017) All Things USB 3.1 And USB Type-C: An Explainer. [Online]. Available: http://www.tomshardware.com/news/usb-31-usb-type-c-refresher,29933.html

[9] (2017) Usb implementors forum. [Online]. Available: http://www.usb.org/home

[10] (2017) USB3300, Hi-Speed USB Host, Device or OTG PHY with ULPI Low Pin Interface. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/00001783C.pdf

[11] (2017) Linaro: Leading collaboration in the arm ecosystem. [Online]. Available: https://www.linaro.org/

[12] (2017) Usbmon - the linux kernel archives. [Online]. Available: https://www.kernel.org/doc/Documentation/usb/usbmon.txt

[13] (2017) GNU Octave. [Online]. Available: gnu.org/software/octave/

[14] S. J. Vaughan-Nichols. (2017) Today's most popular operating systems. [Online]. Available: http://www.zdnet.com/article/todays-most-popular-operating-systems/

[15] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey, "Users really do plug in usb drives they find," *Security and Privacy (SP), 2016 IEEE Symposium on*, 2016.

[16] Cyrus Farivar. (2016) Feds seized 50TB of data from NSA contractor suspected of theft. [Online]. Available: https://arstechnica.com/tech-policy/2016/10/feds-nsa-contractor-stole-at-least-50tb-worth-of-highly-classified-data/

[17] Peter Weber. (2013) How Edward Snowden stole his cache of NSA secrets. [Online]. Available: http://theweek.com/articles/463185/how-edward-snowden-stole-cache-nsa-secrets

[18] Security Research Labs. (2016) USB peripherals can turn against their users. [Online]. Available: https://srlabs.de/bites/usb-peripherals-turn/

[19] Dan Goodin. (2016) Meet PoisonTap, the $5 tool that ransacks password-protected computers. [Online]. Available: https://arstechnica.com/information-technology/2016/11/meet-poisontap-the-5-tool-that-ransacks-password-protected-computers/

[20] R. 'mubix' Fuller. (2016) Snagging creds from locked machines. [Online]. Available: https://room362.com/post/2016/snagging-creds-from-locked-machines/

[21] S. Ramesh. (2017) How to hack passwords using a usb drive. [Online]. Available: https://www.gohacking.com/hack-passwords-using-usb-drive/

[22] D. J. Tian, N. Scaife, A. Bates, K. Butler, and P. Traynor, "Making USB great again with USBFILTER," in *25th USENIX Security Symposium*, 2016, pp. 415–430.

[23] R. Fisk. (2017) The usg is good, not bad. [Online]. Available: https://github.com/robertfisk/USG/