SandUSB: An Installation-Free Sandbox For USB Peripherals

Edwin Lupito Loe*, Hsu-Chun Hsiao*†, Tiffany Hyun-Jin Kim‡, Shao-Chuan Lee*, Shin-Ming Cheng§
*Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
†Research Center for Information Technology Innovation, Academia Sinica, Taiwan
‡HRL Laboratories, USA
§Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taiwan

***Abstract*—This work investigates two emerging attacks—*Human Interface Device (HID) attack* and *Juice Jacking attack*—that leverage USB peripherals, and proposes countermeasures to defend against them. These attacks can be easily reproduced using low-cost IoT prototyping boards (e.g., Raspberry Pi) and can bypass commercial antivirus tools. Although several research prototypes can effectively mitigate Juice Jacking and HID attacks, these prototypes encounter two challenges with respect to deployability: 1) Some require installation on host computers, which is inconvenient and users may lack permission to install software; 2) Some assume cryptographic keys for authentication, but such cryptographic operations may not be supported by legacy USB peripherals and hosts. To address these challenges, this paper presents the design and implementation of SANDUSB, an installation-free and user-controllable security gadget for USB peripherals. Since SANDUSB acts as an intermediary between the USB host and device, SANDUSB can perform efficient scanning and analysis without changing USB devices or hosts. In addition, SANDUSB provides a simple user interface (UI) to control and monitor connected USB devices, enabling users to identify malicious peripherals that masquerade as another type. This UI is complementary to the automatic defensive measures that SANDUSB supports or cryptographic-based authentication. Our evaluation demonstrates that SANDUSB can effectively defend against various USB attacks, including the HID attack and Juice Jacking, using affordable and easily accessible hardware.**

I. INTRODUCTION

USB standards have been the most used interfaces to connect electronic peripherals, including IoT devices [1]. USB standards define several device classes, speeds, and transfer types, with a communications protocol to allow peripherals[1] to communicate with the host device. The most used USB devices today come in the form of USB mass storage (e.g., USB thumb drives), Human Interface Device (HID) devices (e.g., keyboards and mice), and smart devices (e.g., smartphones and IoT devices). With the increase of USB functionality and applications, USB is an attractive attack target. USB thumb drives in particular have become popular attack instruments for spreading computer viruses [5]. Recent attacks masquerade HID devices as USB thumb drives so as to surreptitiously send commands [12], [6]. Also, an adversary can steal or inject mobile data through USB connection during the charging process [13], [10], [14]. Worse yet, a 2016 study shows that most people plug-in untrusted USB drives [22]. Hence, it is imperative to develop effective defense against such USB-exploiting attacks and to raise security awareness for regular users.

Conventional approaches to defend against USB threats heavily rely on the operating system (OS) and anti-virus software to identify anomaly patterns. For example, Microsoft Windows 10 OS disables the USB auto-run, which is commonly used by malware that infects USB drives. However, such defensive measures remain ineffective for HID attacks [8]. Moreover, because mobile devices use USB standards for both charging and data transfer, the adversaries could exploit this charging needs for performing juice jacking attacks to steal private information or even inject malicious code to the victim's devices.

This work studies emerging USB attacks and proposes solutions to defend against them. Although several research prototypes [11], [3], [20], [8], [21] can effectively mitigate Juice Jacking and HID attacks, these prototypes may be inapplicable in some situations due to deployability issues: 1) Some require installation on host computers, which is inconvenient and users may lack permission to install software; 2) Some assume cryptographic keys for authentication, but such cryptographic operations may not be supported by legacy USB peripherals and hosts.

To address these limitations, we design and implement *SandUSB*, an installation-free sandbox for USB peripherals. SANDUSB is a standalone gadget that acts as a layer of protection between USB hosts and devices, thereby avoiding the need of changing USB devices or hosts. When the host and USB devices are connected via SANDUSB, SANDUSB first performs *automatic* defensive measures and then *semi-automatic* defensive measures with the help of human intelligence. The SANDUSB prototype currently supports five automatic defensive measures and can be flexibly extended: 1) USB device blacklist creation, 2) keyboard dynamics analysis, 3) files/settings modification detection, 4) input pattern matching, and 5) simple USB packet analysis. As for the semi-automatic defensive measures, SANDUSB provides a simple GUI to control and monitor connected USB devices, thereby enabling

[1]We will use *USB device* and *peripheral* interchangeably in this work.

users to identify malicious peripherals that masquerade another type. In other words, we leverage users as an alternative means to verify the authenticity of USB devices, so that we can minimize the dependency on cryptographic operations.

We implement SANDUSB prototype using affordable and easy to access hardware. It is built on Raspberry Pi 2 with a 2.8" touchscreen display and relay module, and we design the user interface to help users monitor and control the suspected device. Our evaluation demonstrates that SANDUSB can effectively defend against the HID and Juice Jacking attacks. We envision SANDUSB to be a framework for various USB attacks and can be integrated with existing hardware such as USB hubs to further improve deployability.

II. Background and Related Work

Since mid-1990's, the USB standard has been used to connect computer peripherals and has been evolving to be more complex. Class specification is designed to help programmers when writing the firmware or driver for USB device. Some operating systems prepare class drivers such that devices could be used without the need of installing drivers (i.e., plug and play). This type of devices usually belong to the HID class, such as keyboards, mice, joypad, etc. In some cases, vendors also provide the drivers to support additional features and capabilities for their devices. Currently there are 13 class specifications that have been approved by USB-IF sponsors device working groups.

The USB communication protocol consists of two parties, *host* and *device*, each of which has its own defined responsibilities. A USB host is typically found in computers (e.g., PCs and notebooks). It consists of a USB host controller and a USB root hub. A USB host controller manages all the connected USB devices (e.g., their types and capabilities). When a USB device is connected to a USB host, the USB host detects the device through a process called enumeration, in which the USB host assigns the bus speed, address, and request information (device capabilities, class, etc.) to corresponding devices. The USB device responds to all the USB host requests by sending its capabilities and status to the USB host.

A. Related Work

We review prior work on improving security awareness of USB peripherals, USB attacks, and defenses against USB attacks.

1) Security Awareness of USB Peripherals: Several social experiments on USB observed that many people plugged in unknown USB devices they found [22], [7]. These studies demonstrate that USB drives are an effective attack vector for conducting social engineering attacks. A 2016 study [22] found that among the 297 USB drives dropped on a university campus, 135 (45%) people opened one or more files on the drive that they picked up (plug-in rate), and 290 (98%) of 297 drives were moved from their original locations (found rate). This study indicates that people still lack some security awareness, which this paper aims to improve.

2) USB Attacks: USB attacks can be classified into two scenarios: Malicious USB devices attacking USB hosts, and malicious hosts attacking USB devices.

The plug-and-play feature allows adversaries to masquerade a USB-based device as an HID devices (e.g., keyboard, mouse, joystick). We refer to such attacks as the HID attack. This manipulation could be in the form of pre-programmed device like in USB Rubber Ducky [6] or firmware reprogramming like in BadUSB [12] or through software installation to a rooted device such as Nethunter [16]. The attacked USB host will accept the attacker's payload just like it receives the normal HID device's payload from the user.

Juice Jacking attack exploits the mobile device access permission to steal data or to manipulate the device during charging. As the USB port has a dual functionality of charging and data transferring, a USB device may unintentionally give data access to a malicious host while charging. Moreover, it has been demonstrated that smartphone charging stations can exploit the vulnerabilities of AT commands in Android devices [14] to 1) flash a boot partition and root access, 2) enable adb, and 3) install malicious applications that are unremovable without re-flashing the Android boot partition. Such vulnerabilities have been exploited to bypass the screen lock of the Samsung Galaxy device [15] and affected most Samsung devices before Samsung released a patch to fix these vulnerabilities [17]. Kaspersky Lab reproduced these vulnerabilities and showed that they are conceptually applicable to other vendors' devices [10].

3) Defenses Against USB Attacks: Several systems use virtualization to secure USB. Cinch [3] uses virtualization to secure the USB communication by logically separating the untrusted machine from the protected one, and GoodUSB [20] uses virtualization to defend against BadUSB by enforcing permissions based on user expectations of device functionality. Similar to SANDUSB, Cinch creates a layer of protection between the USB host and peripherals, and GoodUSB uses a USB honeypot mechanism to detect malicious activities. However, Cinch requires the installation of virtual machines on the protected computer, whereas SandUSB defends without requiring any installation on protected machines.

USBFilter [21] implements a firewall feature for USB peripherals, allowing users to manage policies or rules that are similar to firewall rules in a computer network. USBFilter allows users to create USBTables that contain a set of rules for USB peripherals. USBFilter defends against BadUSB attacks by adding a rule that allows a USB packet from trusted mouse and keyboards only, and drops USB packets from untrusted ones. However, USBFilter requires modification on the Linux kernel and is currently only available on the Linux operating system. USBGuard [11] designs a software framework that implements USB device authorization policies to manage and protect USB peripherals. Similar to SANDUSB, USBGuard provides device blacklist creation and device event actions, but only supports Linux OS while SANDUSB is platform-independent. Unlike USBFilter and USBGuard, SANDUSB requires no modification on the host computer and supports all operating systems.

Some solutions are based on cryptography. FlashTrust [19]

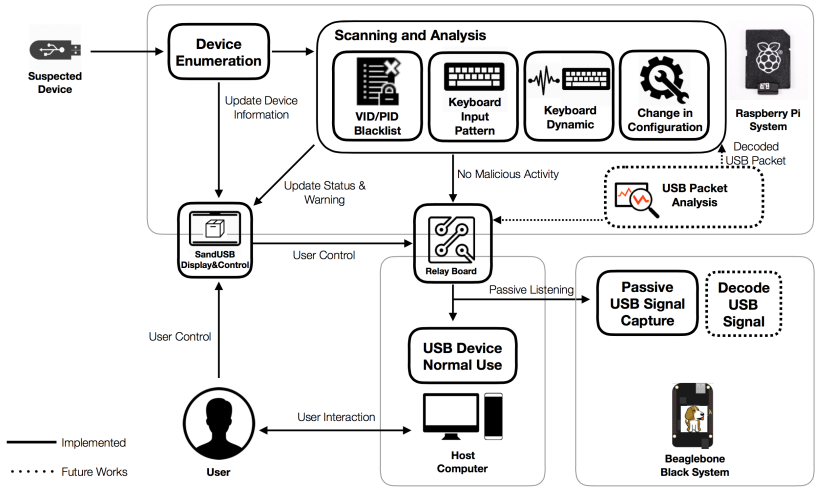

Fig. 1. SandUSB System Architecture.

secures the USB by cryptographically signing its firmware using 2048-bit RSA. This approach can effectively prevent USB firmware reprogramming (e.g., the creation of BadUSB). Our goal is orthogonal: we aim to detect malicious USB hosts and devices regardless how they are created. USBSec [23] adds authentication to protect USB connectivity by adding extensions to the Linux kernel. This approach requires significant amount of modification on the host device, which contradicts with SANDUSB's main principle of requiring no modification or installation on the host device.

USBWall [8] has a similar approach to SANDUSB, using a prototyped embedded computer (Beagle Bone Black) as a sandbox, but USBWall uses a cat5 network cable as a control channel between USBWall and the host computer. Since USBWall is using SSH protocol with port 22, it requires prior configuration or installation on the host device, which conflicts with SandUSB's main design consideration (i.e., no installation or configuration required on the host/device). Another big drawback of USBWall is its performance degradation.

Keyboard dynamics can be used to detect malicious activities such as unhuman typing speed and patterns [4]. SANDUSB also flexibly supports this defense measure.

III. SANDUSB DESIGN

SANDUSB—a sandbox for USB peripherals—sits between the USB host and device, and forwards the USB connection only if no malicious activity is detected in the initial scanning and analysis.

To improve deployability, SANDUSB is designed to meet the following requirements:

1) Building on affordable and easy to access hardware;
2) Users are not required to install anything on the USB host or device;
3) Fast and accurate scanning and analysis processes;
4) Providing a simple UI that enables users to monitor and control the connected USB peripherals.

In addition to our automatic effort to defend systematically, the human factor will also play an important role in SANDUSB to defend effectively against attacks. SANDUSB gives users more control over their USB devices, and more information about the running background processes. User can manually rescan the device and terminate USB device connections. SANDUSB also adds a record keeping feature that is useful in forensic analysis in case of security breach.

Figure 1 illustrates the SANDUSB design. As Figure 1 shows, SANDUSB has three main stages: Device Enumeration, Scanning and Analysis, and Passive USB Packet Analysis.

A. Device Enumeration and Blacklisting

The device enumeration process shows detailed USB device information to the user (USB information profiling). This information includes the vendor ID, product ID, device class, etc. The device class information provides enough knowledge to users to detect USB device masquerading. In other words, device enumeration allows user to easily identify malicious USB peripherals. For example, when a user plugs in a USB Rubber Ducky via SANDUSB and SANDUSB shows that an HID device is attached, he/she may raise suspicion because the USB Rubber Ducky device is not an HID device. In this case, the user can blacklist the identified malicious USB device by simply clicking a button on SANDUSB's UI.

B. Scanning and Analysis

SANDUSB currently supports three defense measures in the scanning and analysis process. For the purpose of implementation, we focus on HID attacks performed by keyboards. SANDUSB can be flexibly extended to support additional defense measures.

- **Keyboard Dynamics Analysis**. This component detects malicious activity by analyzing the keyboard type speed and type pattern, since masqueraded malicious USB device may type inhumanly or with the type speed that is

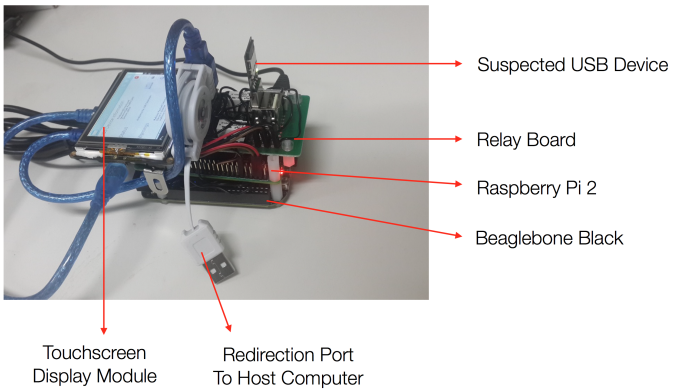

Fig. 2. SANDUSB hardware.

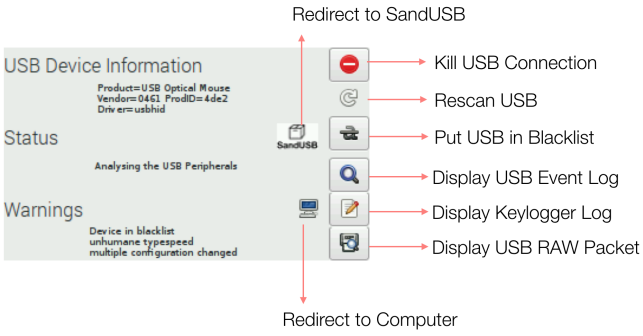

Fig. 3. SandUSB User Interface.

impossible for humans. We note that type speed alone is insufficient in our defense scenario because the attacker can add delays in the payload.

- **Changes in OS Configurations or Files**. Malicious USB device may try to change the OS configuration or steal/inject/modify files on the host when connected. For example, in the BadUSB attack, the malicious device has the ability to change DNS configurations to redirect all traffic to the adversary. SandUSB has the capability to detect changes after the USB device is connected for malicious behavior detection, including zero day attacks in USB peripherals.
- **Malicious Payload Detection**. SANDUSB scans all input payloads and detects any activity that is considered malicious (payload matching), e.g., redirected DNS, run scripts, etc.

C. Passive USB Packet Analysis

After the USB device is considered safe and redirected to the host computer, SANDUSB will still passively analyze the USB device and will redirect the USB device back to SANDUSB when it finds any malicious activity during passive listening. Since forensic is quite important to analyze the attack signature on USB peripherals, SANDUSB maintains logs of USB activities (e.g., time of attachment/detachment), which could be used as evidence of malicious activity. SANDUSB also provides UI such that users can also check the USB packets and logs.

IV. SANDUSB IMPLEMENTATION

This section describes the implementation of SANDUSB.

a) SANDUSB *hardware:* SANDUSB is mainly built on a Raspberry Pi2 Model B, an affordable IoT development board with a Debian-based Linux system. We also use a relay module that consists of 2 TRX-3V Relay ICs that are responsible for relaying USB signals after the SANDUSB's scanning and analysis procedure. This module ensures that the connected USB peripherals are not malicious. Figure 2 shows the hardware implementation.

b) SANDUSB *software:* We used Raspbian as the main OS in SandUSB. Raspbian is optimized for the best performance on Raspberry Pi, and the main programming language is Python 2.7. The graphical UI as shown in Figure 3 is built using the PyGTK library. The GTK+ library is the standard graphical library for the application development in the GNOME Desktop. SANDUSB uses several USB-related command lines (e.g., lsusb, bind, and unbind) to gather information and attach/detach connected USB devices. It also uses the Python libraries for data parsing, command-line handling, and pattern matching. USBMon [24] is used to record the raw USB packets for forensic analysis. SANDUSB also uses the Python gudev library for USB device management to track USB attachment/detachment time. Additionally, SANDUSB uses logkeys [9] to record all input messages during the scanning and analysis processes.

c) SandUSB passive listening: After the relay module redirects the USB signal to the host computer, SANDUSB needs to passively listen to continue protecting the host computer. We implement passive listening by using BeagleBone Black (BBB) Beaglelogic [2], a logic analyzer that is capable of capturing USB traffic at speed up to 100MHz. By using BBB Beaglelogic, the raw packets of USB communication between the host computer and USB peripherals are recorded and transmitted to SANDUSB for passive analysis. SANDUSB will actively mediate USB communication if it detects any malicious activity when analyzing captured USB packets.

V. EVALUATION

We evaluate SANDUSB based on security and deployability.

A. Security

We evaluate whether SANDUSB can successfully defend against various USB attacks. A defense is considered successful if SANDUSB detects the attack and blocks the attacking device from being redirected to the host computer.

We implement both HID and juice jacking attacks using low-cost IoT prototyping boards (e.g., Raspberry Pi) to demonstrate that they can be easily reproduced. In the HID attack implementation, we create an attack platform that allows attackers to attack remotely without pre-loaded attack scripts. Our Juice

TABLE I. COMPARISON OF SANDUSB DEFENSE PROPERTIES

Properties	SandUSB	USBGuard [11]	Cinch [3]	GoodUSB [20]	USBWall [8]	USBFilter [21]
USB Information Profiling	o	x	x	o	o	x
Keyboard Dynamics	o	x	x	x	x	x
Device Blacklisting	o	o	x	o	o	o
Platform Independent	o	x	o	o	x	x
Payload Matching	o	x	o	o	x	o
OS Segmentation	o	x	o	o	x	x
USB Policy	o	o	o	o	o	o
Signature-Based Defenses	x	x	o	x	x	x
Authentication and Encryption	x	x	o	x	x	x
Driver Vulnerabilities Defense	x	x	o	x	x	x
Limiting USB Device Functionalities	x	x	x	x	x	o

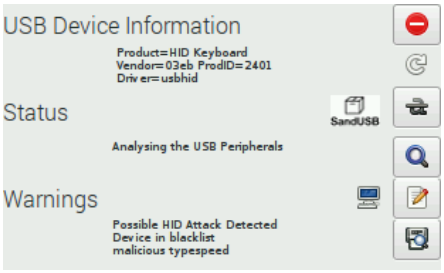

Fig. 4. SandUSB analysis results of an HID attack.

TABLE II. SANDUSB SPEED PERFORMANCE BENCHMARK USING XBENCH. A HIGHER SCORE INDICATES A FASTER TRANSFER SPEED.

	Without SandUSB	With SandUSB
Result xbench Score	9.04	8.04
Sequential xbench Score	17.43	17.29
Uncached Write (4K blocks)	8.13 MB/s	8.13 MB/s
Uncached Write (256K blocks)	6.72 MB/s	6.77 MB/s
Uncached Read (4K blocks)	6.72 MB/s	6.68 MB/s
Uncached Read (256K blocks)	19.22 MB/s	17.95 MB/s
Random xbench Score	6.10	5.24
Uncached Write (4K blocks)	0.60 MB/s	0.57 MB/s
Uncached Write (256K blocks)	0.69 MB/s	0.57 MB/s
Uncached Read (4K blocks)	4.34 MB/s	4.16 MB/s
Uncached Read (256K blocks)	18.34 MB/s	18.08 MB/s

jacking attack script can scan all device files and retrieve secret files. We also test SANDUSB against existing attacking tools such as USB Rubber Ducky and NetHunter.

1) Defending Against Hid Attacks: Figure 4 shows the analysis results when USB Rubber Ducky was connected to SANDUSB. (The results for NetHunter and our HID attack are similar.) Three warnings were triggered. First, a possible HID attack is detected because an additional HID device is detected. The user can also visually confirm this mismatch by checking the device information displayed on the screen. The second warning informs users that the connected device has previously been put on SANDUSB's blacklist. SANDUSB also displays a warning message that the device has a malicious type-speed or malicious payload based on the detection after the USB device is plugged-in.

B. Defense Against Juice Jacking Attacks

SANDUSB acts as a sandbox and thus can logically separate a USB device from a malicious host. That is, the user can simply charge his or her device via SANDUSB without the need to redirect its connection to the host computer. Alongside logically separating the malicious host and USB device, SANDUSB can also help the user detect the juice jacking attack after the USB signal is redirected to the host computer. SANDUSB can analyze or display the data transfer status (e.g., the real-time signal status in d+ and d- USB data pins) between the USB host and device during passive listening. The user can then detect juice jacking by monitoring the d+ and d- signal status during the charging period.

1) Chance to Detect Zero-Day Attacks: Since SANDUSB acts as a sandbox for USB peripherals, SANDUSB can immediately detect malicious activities initiated by the USB device, simply by monitoring the modification of configurations and files. SANDUSB has one purpose similar to the sandbox in network security, which is to defend against zero-day (i.e., unidentified) attacks by analyzing the behavior and event inside the SANDUSB system. More specifically, SANDUSB runs a script to detect any modifications on configurations or files in its systems, increasing the chance to detect zero-day attacks.

2) SandUSB Defense Properties: We compare SANDUSB's defense properties with the other related work on securing USB communications. The defense properties include USB device blacklisting, keyboard dynamics, USB device events, HID payloads matching, keyboard input log, and USB packet log. Table I highlights the SandUSB defense properties compared to USBGuard, Cinch, GoodUSB, USBWall, and USBFilter.

C. Deployability

1) USB Speed Performance: To evaluate the performance overhead, we measure the USB speed with and without SANDUSB. Specifically, a USB thumb drive is connected to the host computer, and we measure the transfer speed by using xbench [18]. The comparison result is shown in Table II. SANDUSB only increases the xbench score by one, indicating acceptable overhead.

Since SANDUSB passively listens to the USB connection between the USB host and device, it adds no additional operational overhead to the host after redirection. However, there may be timing issues when SandUSB detects some malicious activity during passive listening, in which case the malicious actions may already be launched to the host computer. The delay time is approximately equal to the signal processing and decode time plus the analysis time. A goal of our future work is to minimize this delay time through efficient signal processing and decoding.

2) SANDUSB *Scalability:* SANDUSB could support more than one USB device by using a USB hub, through which it can scan all connected USB devices and store the retrieved information inside its data structure. The user could also modify the representation of information in SANDUSB by simply modifying SANDUSB's source code. Currently, SANDUSB does not support USB 3.0 standard due to hardware limitations of Raspberry Pi 2 (which only supports the 2.0 speed). The connected USB 3.0 device is still usable since USB 3.0 is backward-compatible and can be processed at the 2.0 speed. SANDUSB might be able to support USB 3.0 in the future through hardware upgrade.

3) Cost of SANDUSB*:* Cost was an important factor while designing SANDUSB. We built the SANDUSB prototype on easily accessible hardware, such as Raspberry Pi 2 and Beaglebone Black, and the touchscreen display module can be replaced selectively. Additionally, the hardware of the relay module is very simple and affordable. The overall cost of the SANDUSB prototype is $138, and the cost of SANDUSB could be further scaled down for production.

VI. CONCLUSION AND FUTURE WORK

In this work, we study two emerging attacks that leverage USB: HID and Juice Jacking attacks. We build a platform to commence HID attacks remotely, and use Raspberry Pi as a power adapter to commence Juice Jacking attacks. These attacks are proven to be very effective and dangerous, as the instruments for launching these types of attack are easily accessible and require little technical knowledge to commence.

Our solution, SANDUSB, defends against these attacks. As a sandbox for USB peripherals, SANDUSB acts as a layer of protection between the USB host and the device. More specifically, SANDUSB is responsible for scanning and analyzing the USB device before usage, and it has some built-in defensive measures to detect malicious activities of connected USB peripherals. SANDUSB provides the user with additional information and control of the connected USB device, but user awareness remains as an important factor for our defensive measures to be effective.

We aim to improve SANDUSB along several directions. For example, we plan to extend SANDUSB into a robust and effective defensive framework for protecting USB peripherals against various attacks, in addition to HID and juice jacking attacks. We would also like to deploy USB policies similar to Intrusion Detection/Prevention Systems. We also want to reduce the overhead of our software-based relay module. Finally, we envision that SANDUSB can be integrated with existing hardware as an IoT product such as a secure USB hub. Such a secure USB hub would be able to automatically scan, analyze, redirect, or block a USB device that is connected through the secure USB hub.

ACKNOWLEDGMENTS

This work was supported in part by Taiwan Information Security Center (TWISC), Academia Sinica, and Ministry of Science and Technology, Taiwan, under the grants MOST 104-2218-E-001-002 and 104-2923-E-011-006-MY2.

REFERENCES

[1] Two decades of "plug and play": How USB became the most successful interface in the history of computing. http://www.intel.com/content/www/us/en/standards/usb-two-decades-of-plug-and-play-article.html, Last accessed 16 July 2016.

[2] K. Abhishek. Beaglebone Black Beaglelogic. https://github.com/abhishek-kakkar/BeagleLogic, Last accessed 16 July 2016.

[3] S. Angel, R. S. Wahby, M. Howald, J. B. Leners, M. Spilo, Z. Sun, A. J. Blumberg, and M. Walfish. Defending against Malicious Peripherals with Cinch. In *USENIX Security*, 2016.

[4] F. A. Barbhuiya, T. Saikia, and S. Nandi. An anomaly based approach for HID attack detection using keystroke dynamics. In *Cyberspace Safety and Security*, pages 139–152. Springer, 2012.

[5] A. Gostev. Kaspersky security bulletin. *Statistics*, pages 68–73, 2008.

[6] Hak5. Usb rubber ducky. http://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe?variant=353378649, Last accessed 16 July 2016.

[7] J. R. Jacobs. *Measuring the effectiveness of the USB flash drive as a vector for social engineering attacks on commercial and residential computer systems*. PhD thesis, Embry-Riddle Aeronautical University, 2011.

[8] M. Kang. *USBWall: A Novel Security Mechanism to Protect Against Maliciously Reprogrammed USB Devices*. PhD thesis, University of Kansas, 2015.

[9] kernc. logkeys - a gnu/linux keylogger. https://github.com/kernc/logkeys, Last accessed 16 July 2016.

[10] A. Komarov. Wired Mobile Charging—Is it Safe?, 2016. https://securelist.com/blog/mobile/74804/wired-mobile-charging-is-it-safe/, Last accessed 16 July 2016.

[11] D. Kopeek. USB Guard: USB device authorization policies, 2016. https://dkopecek.github.io/usbguard/, Last accessed 16 July 2016.

[12] K. Nohl and J. Lell. BadUSB—On accessories that turn evil. *Black Hat USA*, 2014.

[13] W. of Sheep. Juice Jacking Attack: USB as a charge port. http://www.wallofsheep.com/pages/juice, Last accessed 16 July 2016.

[14] A. Pereira, M. Correia, and P. Brandão. USB connection vulnerabilities on Android smartphones: default and vendors' customizations. In *IFIP International Conference on Communications and Multimedia Security*, pages 19–32. Springer, 2014.

[15] A. F. Roberto Paleari. Samsung Galaxy phone lock screen bypass, 2016. https://github.com/ud2/advisories/tree/master/android/samsung/nocve-2016-0004, Last accessed 16 July 2016.

[16] O. Security. Kali Linux Nethunter. https://www.kali.org/kali-linux-nethunter/, Last accessed 16 July 2016.

[17] S. M. Security. SVE-2015-5301: Disable AT Command via USB with secured lockscreen, 2016. http://security.samsungmobile.com/smrupdate.html#SMR-JUN-2016, Last accessed 16 July 2016.

[18] S. Software. XBench: Benchmarking tools on Mac, 2008. http://www.xbench.com/, Last accessed 16 July 2016.

[19] K. Solutions. Kanguru Flash, 2016.

[20] D. J. Tian, A. Bates, and K. Butler. Defending Against Malicious USB Firmware with GoodUSB. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 261–270. ACM, 2015.

[21] D. J. Tian, N. Scaife, A. Bates, K. Butler, and P. Traynor. Making USB Great Again with USBFILTER. In *USENIX Security*, 2016.

[22] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey. Users Really Do Plug in USB Drives They Find. In *IEEE Symposium on Security and Privacy*, 2016.

[23] Z. Wang, R. Johnson, and A. Stavrou. Attestation & Authentication for USB Communications. In *IEEE International Conference on Software Security and Reliability Companion*, 2012.

[24] P. Zaitcev. The usbmon: USB monitoring framework. In *Linux Symposium*, 2005.