Bypassing USB Security with HERMES: A Novel Approach to Dynamic HIDs Simulation

Yu-Wei Hsu, Fu-Hau Hsu, Tzu-Hsun Yuan, Keyuan Wang, Jian-Hung Huang, Hao-Jyun Wang, Jian-Xin Chen, Min-Hao Wu*

***Abstract*—The prevalence of Universal Serial Bus (USB) devices has brought about a pressing need for heightened security measures, owing to the rise of sophisticated threats such as BadUSB. Introduced by security researchers, this malware variant modified USB firmware and thus enables it to exploit vulnerabilities across diverse systems. Existing mitigation strategies, predominantly relying on allowlisting solutions, have proven inadequate against the evolving and persistent threat landscape. HERMES, as introduced in this study, represents a covert USB attack method designed to replicate plug-and-play device characteristics in real-time, thus evade traditional firewall defenses. Contrary to contemporary cybersecurity frameworks, which focus on device authentication based on attributes such as manufacturer details, serial numbers, etc., HERMES exploits dynamic feature replication to bypass firewall restrictions undetected. By spoofing as legitimate devices, HERMES infiltrates into target systems, facilitating the execution of malicious scripts instantaneously. This paper reveals the vulnerabilities inherent in contemporary USB security paradigms and underscores the urgency for innovative countermeasures. Beyond raising awareness of potential risks, we look forward to suggest serving HERMES as a novel approach for validating the effectiveness of various defense methods, contributing to the ongoing efforts against emerging BadUSB vulnerabilities and USB-based threats.**

***Index Terms*—BadUSB, Human Interface Device, Universal Serial Bus, Firmware, Security, Emulation, USB Enumeration, Testing Platform, Spoofing, Firewall**

I. Introduction

PRIOR to the introduction of USB (Universal Serial Bus) in the 1990s, connecting peripherals to personal computers often involved in a variety of connectors for specific interfaces and platform, including serial and parallel ports, PS/2, SCSI, game ports, ADB and others. Obviously, This caused variant issues of convenience; especially most of these connectors were usually not hot pluggable and also needed manually configuration options. The widespread adoption of USB addressed many of these shortcomings by introducing a universal standard for peripheral connectivity. Unlike earlier connectors, USB peripherals have plug-and-play functionality now and can be easily plugged into a host, which automatically handling hardware configuration. However, while USB's plug-and-play functionality offers convenience, it is simultaneously its greatest and most dangerous property. The ease with which USB devices can be connected to a host increases the potential for unauthorized access and malware intrusion [1].

Throughout the early 21st century, adversaries have been using USB peripherals to launch a variety of cyberattacks, exploiting the vulnerabilities of these devices [2]. Based on in 2011 research proposed by D. V. Pham, A. Syed and M. N. Halgamuge, most USB attacks in the past relied on USB device storage, malware associated with Autoplay USB drives, and maliciously modified USB drivers designed to exploit buffer overflow vulnerabilities, primarily for the privilege escalation purposes [2], [3].

Nowadays, the landscape of cybersecurity has witnessed the emergence of novel threats, with many novel and more sophisticated attacks based on utilizing malicious USB peripherals looming large. They typically involve non-USB human interface devices (HIDs) that pose as a keyboard or mouse whose firmware has been modified [2]. Following the surge of new HIDs threats, the research according to K. Nohl, S. Krißler and J. Lell at Black Hat Conference in 2014, revealed this new attack method named BadUSB [4]. By exploiting the inherent techniques of the feature that the firmware of USB device can be modified and updated, the malicious modifications of firmware by adversaries embedded with SoC (System on Chip) or MCUs (microcontrollers) can turn the benign USB device into a weapon for attackers. Thus, BadUSB can spoof as legitimate peripherals, facilitating the execution of malicious scripts upon connection to unsuspecting systems. By implementing the execution of these attacks through a straightforward scripting language, allowing individuals to craft payloads and capable of altering system configurations, creating backdoors, retrieving data, initiating reverse shells, deploying malware, or conducting any other actions feasible with physical access. Furthermore, these actions can be automated and executed within a matter of seconds.

In response to the pervasive threat posed by BadUSB and similar attacks [5], [6], the cybersecurity researchers and community have developed various defense mechanisms aimed at safeguarding systems against USB-based threats. These defense strategies predominantly rely on feature-based firewalls, such as USBGuard [7], SolarWinds USB Defender [21], USB Keyboard Guard [20], USB Waechter [10], Guardian - USB Whitelisting Script [13], etc [1], [8], [9], [11], [12]. These solutions operated by analyzing the USB device characteristics and establishing blocklists or allowlists to mitigate the risk of unauthorized device access.

Nevertheless, the arms race between attackers [23], [24] and defenders in the cybersecurity landscape demands continuous innovation. In this context, HERMES, proposed in this research, can emerge as a formidable offensive tool in the arsenal of attackers. HERMES could exploit built-in USB Gadget Mod emulation to replicate USB device characteristics, including all descriptors, in real-time. By meticulously cloning target USB devices, HERMES can bypass traditional feature-based firewalls, enabling it to legitimately execute malicious code and instantly launch BadUSB attack.

The distinctive feature of HERMES lies in its ability to accurately simulate all aspects of target USB devices, enabling it to bypass USB firewall and circumvent existing defense mechanisms with unprecedented precision. Contrary to conventional security solutions, which rely on static device attributes to detect against BadUSB attack, HERMES dynamically adapts mitigation approach of USB by inserting into existing firewall allowlisted devices and instantly replicating their characteristics. This action results in that the firewall misidentifies them as allowlisted devices, thus enabling HERMES to bypass defense mechanisms of most USB, which can pose a significant challenge to USB security.

In conclusion, our contributions to this research and methods of HERMES can be classified in four points. First, HERMES that we developed can pass through traditional USB firewalls effectively by replicating descriptors of USB devices. Secondly, HERMES provides a simple plug-and-play platform using the Pi Zero 2 W as our attacking device. Upon insertion of the USB that is pre-registered in the target system, initiating the HIDs attack automatically and streamlining the intrusion process. Subsequently, we have conducted rigorous validation tests to assess the effectiveness and robustness of HERMES against various USB mitigation methods. Our experiments yielded successful results compared to traditional BadUSB attack method, affirming the robustness of HERMES's design in real-world scenarios. Finally, by proposing HERMES, we look forward to serve HERMES as a better approach for future USB security researchers to validate the effectiveness of mitigation or defense methods on BadUSB attack.

II. Background

The USB-based threats, or say Human Interface Device (HID) attack through USB devices, involves fake peripheral devices using like USB Rubber Ducky, Raspberry Pi or any evaluation boards to conceal malicious programs. These malicious programs would be directly loaded and executed, causing an instantaneous attack. Instead of relying on user interaction, HID attacks capitalize on the plug-and-play functionality of USB devices, exploiting the seamless integration between peripherals and host. This method of attack would bypass traditional security measures, often without triggering alarms or detection mechanisms. For instance, BadUSB attack represents a typical example of the USB-based threats. Because of the features of BadUSB attack, not like the traditional USB threats which are based on storing in USB flash memory, exploits conducted by malware are from the modified firmware or ROM (Read Only Memory). By interacting with the benign host according to the USB protocol and USB enumeration, the BadUSB device would process all malicious steps called BadUSB Enumeration like in the Fig. 1. Due to the lack of strict detection measures for such devices by computers, which typically only identify device types superficially, only by tampering with device feedback information, adversaries becomes relatively easy to trick the computer into recognizing another device as an HID device. The host then mounts the driver based on the provided information, thereby allowing the device to perform various operations on the computer like initiating a series of keystrokes which issue commands to download malware on the remote site, open a reverse shell or execute malicious scripts legitimately. Particularly with the advent and the following prevalence of plug-and-play interfaces like USB and Bluetooth, HID attacks such as BadUSB attacks have potential risk become a primary means of attack.

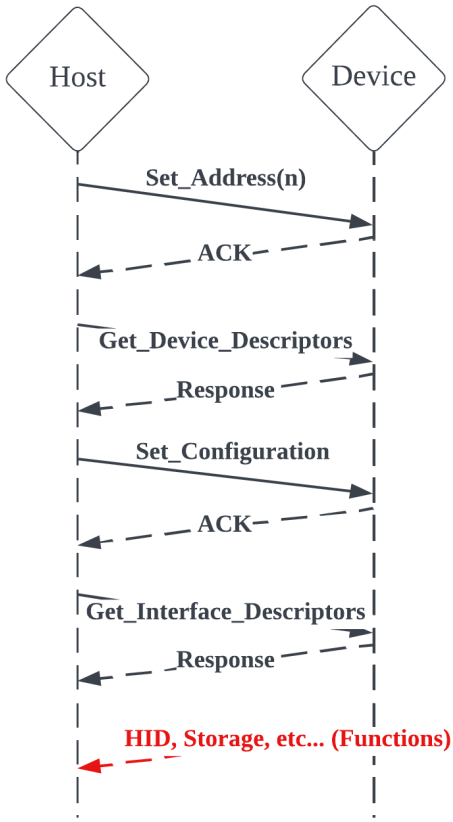

Fig. 1. BadUSB Enumeration that create communication between host-side and device-side. The red dotted line, in the traditional BadUSB attack scenario, illustrates that when the malicious device requests additional interfaces (functions), the host will also take it as correspondent devices.

A. USB Controller

The corresponding structure between the operating system kernel and USB is like Server/Client relationship. Through the USB host controller (UHC), the commands occur in the host would request USB device controller (UDC), serving as an intermediary between the device and the host, to respond information. Moreover, the UHC is responsible for managing USB, including handling USB devices, USB enumeration, and descriptors; essentially, UHC enables communication with all USB devices. On the other hand, a UDC can enumerate itself to the host, providing essential information like USB class, Vendor ID (VID), Product ID (PID), driver version, power requirements, and protocol within the device controller's descriptor. The host must retrieve this information in descriptors to establish proper communication with the USB device.

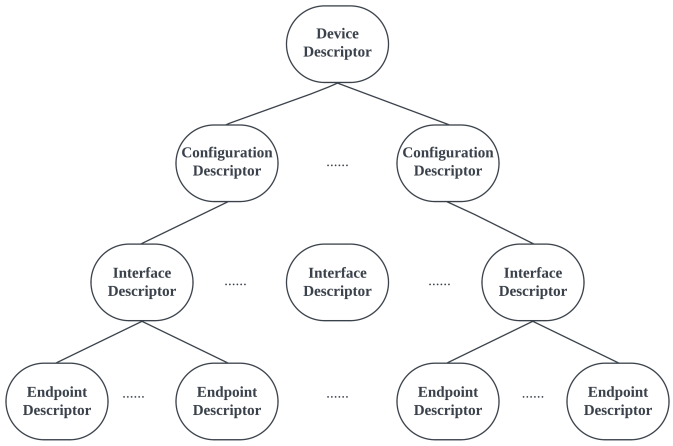

Fig. 2. Tree graph of different descriptors' relations.
A USB device would only have one device descriptor, by comparison with that USB device have multiple configuration, interface, endpoint descriptors. Depends on what active configuration descriptor is, one or more interface and endpoint descriptors would be acquired by the UHC in the USB enumeration process. Thus, for different type of device, the host will assign applicable kernel drivers based on the specific configuration descriptor to USB device to load.

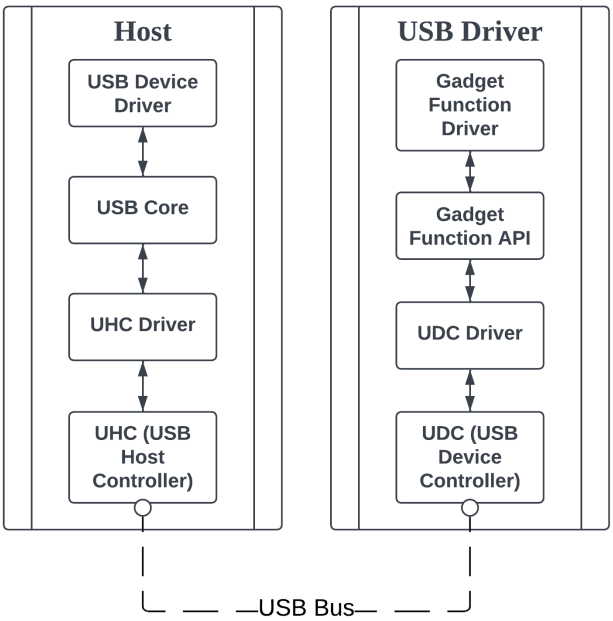

Fig. 3. The architecture of Linux kernel and USB driver.

B. USB Enumeration

When a USB device needs to connect to a host, the host must know the device's type and requirements. Therefore, it is necessary to inquire about certain aspects of the device, such as configurations, interfaces etc., by USB enumeration. Following a series of requests and responses, if the host allow that the device can be connected, it will proceed to install the essential kernel drivers according to the interfaces the device asserts, thereby utilizing the function of the device correctly, completing the USB enumeration. However, indicated by the red dotted line in Fig. 1, in the traditional BadUSB attack, as long as the malicious device requests additional interfaces, enabling the device to legitimately spoofing and execute malicious code on the system [17].

In detail, upon insertion of a USB device into the host machine, the UHC undertakes the task of identifying its presence and determining its speed by detecting voltage variation on the data pins. After confirm the connected device in the physical layer, Enumeration then begins with the Get_Device_Descriptors command, whereby the host requests the device for its identifying information including manufacturer, Vendor ID (VID), Product ID (PID), and serial number. Following this, shown in Fig. 1, the UHC initiates a reset on the USB device and sends a Set_Address command to ensure their future communication; the host would thus request Get_Device_Descriptors again to reaffirm if new address is assigned properly to UDC. Subsequently, a Set_Configuration request is issued to retrieve all available configurations within the USB device. Notably, a USB device would only have one device descriptor, but a device descriptor may offer multiple configuration descriptors; although only one can be active at any given time [25].

After Set_Configuration is done, the UHC through the Get_Interface_Descriptors request acquired one or more interface descriptors under the specific configuration descriptor, which UDC had responded. Additionally, These interface descriptors contain multiple endpoint descriptors and essentially represent distinct functional entities served by different drivers within the operating system (OS). Finally, after the completion of the Get_Interface_Descriptors command, kernel drivers known as interfaces are loaded as a representative of device and allow the USB device to operate; USB device will respond class-specific subsets of the USB functions like HID and Storage to the UHC, which are established and activated accordingly [25].

C. USB On-The-Go (OTG)

USB On-The-Go (OTG) was released in November 2000 with a version number of 0.7. It serves as a supplementary specification for USB 2.0, enabling both host and device functionalities if OTG support is present on both ends, like in Fig. 3. Especially, this means that the host can act as a device and vice versa. One of the most known applications is the USB interface on mobile phones. Furthermore, OTG has been available for USB controller since version 2.6 of the Linux kernel in the drivers. The SoC (System on Chip) or MCUs (microcontrollers) like BCM, PXA, OMAP, and AT91, can emulate both host and device chips including HID, Mass Storage, Webcam (V4L2), Ethernet, and others. Otherwise, some MCUs don't support the OTG or have no host controller interface or USB controller, which only can act as embedded host, are not included in this description.

D. USB Gadget Mode

The main function of USB Gadget Mode is to emulate a standard USB device using Linux kernel. This kernel module

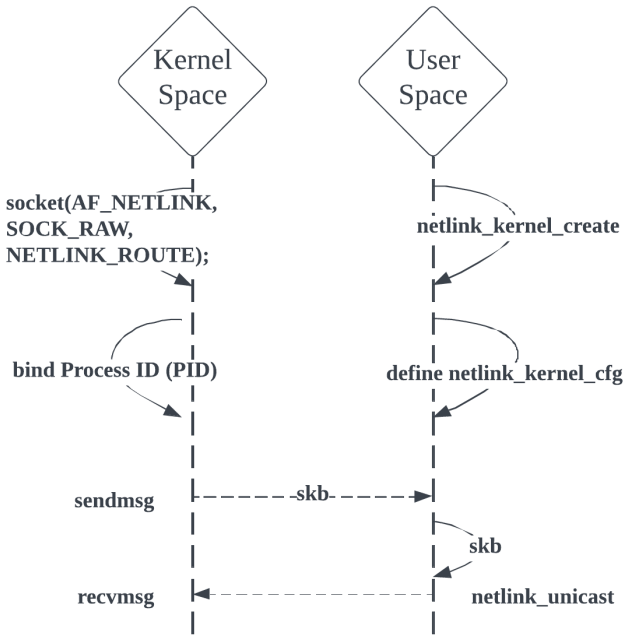

Fig. 4. Native Linux Netlink API creates Netlink socket and establishes communication on kernel-side (kernel space) or on user-side (user space).

integrates drivers for multiple SoC or MCUs such as BCM for Raspberry Pi Zero 2 W, PXA for Intel, OMAP for TI, AT91 for Atmel, and others. As long as the platform has USB controllers with supported drivers and supports OTG natively, it's possible to use Linux Gadget Mod to emulate USB devices; these system architecture of the Linux host and device sides are illustrated in Fig. 3.

In connection with this feature of Gadget Mode, HERMES can simulate USB devices through a combination of the Gadget Mode within the Linux system kernel and hardware OTG. After exploiting the OTG to sniffing the event packets, configuring Gadget Mode and constructing Gadget device, HERMES then simulates these allowlisted USB devices based on their descriptors that are originally allowlisted. Finally, this spoofing enables HERMES to bypass USB firewalls such as USBGuard and USBFilter by issuing built-in exploit script to the system.

E. Netlink

USB hotplugging is widely used for HID devices, USB flash drives, mobile phones, network adapters with USB interfaces, etc. The Netlink socket is the vital mechanism for hotplugging by impletment the inter-process communication like detection, monitoring and processing of UEvent (or called hotplug events) between OS kernel in kernel space and corresponding device interfaces in user space.

Linux system kernel usually uses Udev to manage UEvent; by utilizing Netlink to monitor Udev packets and then parsing the strings within these packets, the host can further identify which device it is, thus completing the asynchronous communication and logical control to the user space.

Moreover, shown in Fig. 4, Netlink supports the process-to-process multicast. A process on the kernel space can send an event or message to Netlink for communicating with any number processes on the user space. In contrast, a process can also broadcast an event to Netlink in the user space, so some kernel modules or applications can receive this message. This provides an efficient mechanism for asynchronous communication of messages from the kernel to user space.

III. Related Work

Due to the development of USB in the past 30 years, various attack methods have emerged. Because of the plug-and-play functionality, the people who work or have permissions to access to critical infrastructure, sensitive files, or various data centers and information systems may overlook USB's inherent security vulnerabilities owing to its convenience. In particular, according to a 2016 study by M. Tischer et al. people have a 45% to 98% success rate of inserting an USB device with malicious payload into a computer and triggering a USB-based attack [15]. In the 2018 paper "SoK: "Plug & Pray" Today - Understanding USB Insecurity in Versions 1 Through C" by J. Tian et al, the USB attack vectors are systematically categorized into USB-based attacks against the human layer, application layer, transport layer down to the physical layer and different communication layers. For instance, BadUSB is considered as "The abuse of Transport Layer", "Protocol Masquerading" and "Rather exploiting protocol's inherent to augment functionality that users would not think to look for" [25]. If only specific feature-based defense mechanisms are implemented against BadUSB, there is still a high probability that BadUSB will be able to carry out successful cross-layer attacks. The GoodUSB allowlist approach proposed by D. J. Tian, A. Bates, and K. Butler [17]; USBFilter packet firewall invented by D. J. Tian et al. [1]; Cinch virtualization isolation mechanism proposed by S. Angel et al. [26], SandUSB [28]; a virtualization method SandUSB identical to Cinch, proposed by E. L. Loe et al; and the study by G. Hernandez et al. called FirmUSB applied symbolic execution [14], are some of the latest USB defense solutions. USB defense solutions, especially against BadUSB, are through the application layer (allowlist), the transport layer (signed firmware, packet firewall, virtualization isolation) or the physical layer (Secure channel) to prevent malicious peripherals. These advance USB defense solutions based on their similar characteristics can be slightly classified into three categories. First, GoodUSB and USBFilter are especially based on allowlist for access control to prevent BadUSB. Secondly, GoodUSB, Cinch and SandUSB are all using virtualization to isolate malicious peripheral devices. Moreover, Cinch and SandUSB use virtualization, constructing a protection layer to logically separate untrusted devices from malicious activities, to petect the communication of host with external devices. Compared to them, GoodUSB is placing virtualization isolation after the allowlist. As long as the allowlist of GoodUSB don't recognize the access device as benign, GoodUSB will redirect the malicious device into virtualized honeypot to keep the host safe. Thirdly, FirmUSB and Ironkey [31] are utilized firmware validation methods to block BadUSB attack.

1) GoodUSB does not normally disable the HID interface for all devices because of the availability tradeoffs for specific devices that need to use the HID interface (e.g., USB headsets, because of their own functionality, must inevitably need to control the volume of the internal speaker and must not disable the HID interface). Under normal circumstances, GoodUSB can avoid further threat of malicious script execution by manually confirming the type of HID device plugged in; if it is a traditional BadUSB attack, the malicious device will be detected by GoodUSB's allowlist and redirected to the virtualized honeypot and stay for the security analysts doing further digital forensics. However, because of GoodUSB's allowlist feature, HERMES let the USB connect the HID device, which had already connected to the target computer, thus can still copy its descriptors and bypass the GoodUSB allowlist mechanism to achieve the BadUSB attack.
2) USBFilter [1] implements a stack-based firewall for USB peripheral devices and allows users to configure policies to constrain permissible protocol activities. In addition, USBFilter can also create USBTables that contain rulesets for USB peripheral devices. Similar to netfilter and iptables, USBTables would not only block untrusted devices by only accepting HID devices such as mice and keyboards in the allowlist, it will also match USB buses and ports to filter traffic to avoid BadUSB. However, USBFilter can only run on Linux operating systems and requires modifications to the Linux kernel. What's more, the USBFilter method mainly prevents BadUSB attacks through the allowlist, which is roughly the same as GoodUSB. As long as advasaries connect the HID devices that are already in the allowlist to our HERMES, the maliciously modified USB devices can still copy their descriptors and insert them into the target computer to complete the BadUSB attack.
3) Alternatively, Kingston Ironkey [31] uses signed firmware to solve the BadUSB problem, resulting in increased attack complexity, but signed firmware is not only expensive, it can also be compromised in the event that the device manufacturer is attacked, the device manufacturer is not trustworthy, or the legally signed key is insecure. Because the absence of a defense layer on the host plus the compromised signed firmware, normal BadUSB attacks and HERMES is still able to successfully exploit the USB transmission process and successfully get access.

In 2014, the USB Implementors Forum made it clear that security falls outside the scope of the USB specification [22]. In an official statement, the USB-IF asserted that security concerns were not a legitimate concern, as "In order for a USB device to be corrupted, the offender would need to have physical access to the USB device." They emphasized that the responsibility for security lies with both consumers of USB products and the original equipment manufacturers (OEMs) [25], stating:

1) "OEMs decide whether or not to implement these [security] capabilities in their products."
2) "Consumers should always ensure their devices are from a trusted source and that only trusted sources interact with their devices." [25]

This lack of a unified security solution between OEMs has indirectly led to the fact that today's defenses against specific attack vector are being bypassed and exploited one by one by newer USB-based attack techniques. We have yet to see new integrated USB security solutions to defend against cross-layer attacks [15] such as normal BadUSB attack and HERMES in this study.

IV. System Design

This section will present the overall architecture of the system, including the functions, interrelations, and workflows of each component. The system aims to provide a framework capable of identifying, simulating, and exploiting USB devices for specific security testing or attack purposes. The four main components include monitoring and identification, information replication, USB construction, and attack. The functionality and interrelation of each component will be systematically introduced.

Before delving into the system's overall architecture, it is necessary to first establish a high-level security environment. The objective is to enable HERMES to effectively operate against this environment.

A. Defense Scenarios

This section delves into three levels of defense mechanisms against USB threats. Firstly, in environments equipped with IDS/IPS systems capable of detecting malicious USB devices, prevalent in contemporary firewall designs [1], [7]–[13], [20], [21], minimal user intervention is required. These systems autonomously identify anomalies in device descriptors, maintaining blocklists of VendorID and ProductID reported as malicious. Furthermore, some systems restrict the use of USB composite types commonly exploited by BadUSB. Secondly, feature-based firewalls employing allowlisting techniques represent a more sophisticated security approach, requiring user authorization to connect USB. Only authenticated devices are permitted to interface with the system. These firewalls scrutinize USB devices during enumeration, cross-referencing them with a allowlist of approved devices. Thirdly, the most stringent environments mandate user classification of expected functionalities for every connected device. Any deviations from predefined restrictions are flagged as potentially malicious. Additionally, some systems monitor input buses, employing keyword filtering and machine learning algorithms to analyze device behaviors.

In an advanced security environment, only basic input devices such as a mouse and a keyboard, which are pre-registered in the host allowlist, are permitted for USB connectivity. Upon connection of either the mouse or keyboard, rigorous verification processes are initiated to validate all aspects of the device attributes, configurations, functionalities, and the report descriptor. These verifications aim to thwart attacks involving spoofed VendorID and ProductID, while also scrutinizing the

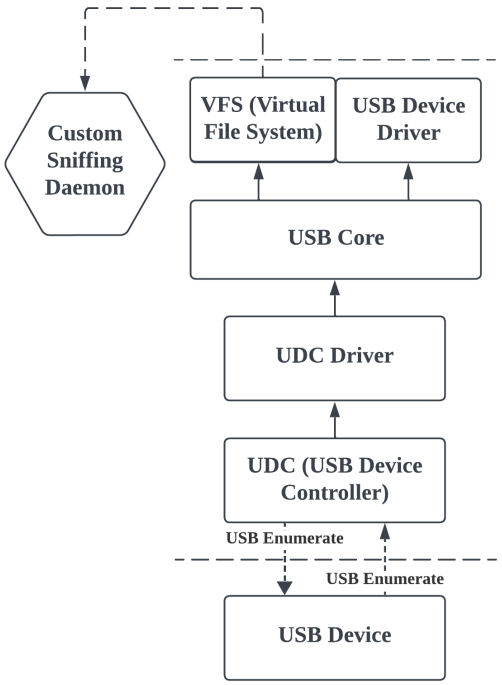

Fig. 5. Key components for sniffing and identification feature of HERMES (between Subsystems of Sniffing Daemon and Virtual File System).

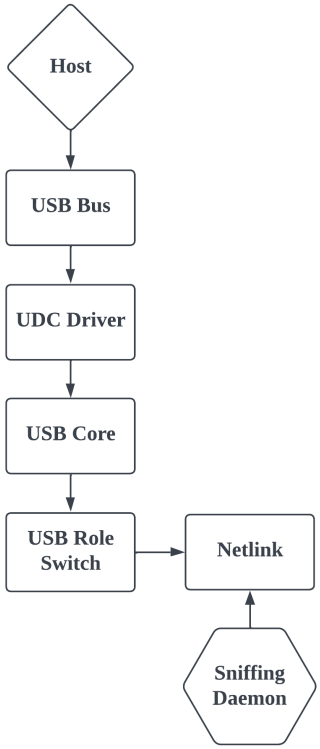

Fig. 6. Key components for sniffing and identification feature of HERMES (between Subsystems of Sniffing Daemon and Netlink).

driver loaded by the host for further authentication. Furthermore, continuous monitoring of every report transmitted from the Human Interface Device (HID) is conducted to detect any signs of malicious behavior. Upon detection of such behavior, immediate ejection of the offending device is enforced to prevent any potential security breaches.

B. Sniffing and Identification

The sniffing and identification of USB devices involve multiple key components that work together to ensure real-time detection and handling of USB devices or host insertion events, enabling HERMES to achieve plug-and-play functionality. At the core of the monitoring and identification process is the Sniffing Daemon, comprised of three subsystems: the Action Decision Maker, the Device Information Extractor, and the Attributes Database.

The Action Decision Maker continually sniffs VFS(virtual File System) paths and Netlink messages to determine whether HERMES is connected to a USB device or a host. When a USB device is connected to HERMES, the USB core subsystem registers the device under /sys/bus/usb/devices/ after loading the USB driver using VFS. Conversely, when a host is connected to HERMES, the USB core invokes the USB Role Switch to initiate USB Gadget mode, enabling HERMES to function as a USB device, and notifies the daemon in user space via Netlink. Based on the determination, it decides the next course of action. If it detects a USB device, it invokes the Device Information Extractor; if it identifies a host, it executes the BuildUSB function to simulate a USB device. Once the Device Information Extractor conducts its task, it extracts descriptor information from the USB device, including the Device descriptor, Configuration descriptor, Interface descriptor, Endpoint descriptor, and the report descriptor for HID devices. It also determines whether the device has an HID keyboard interface; if not, it adds a predefined HID keyboard interface to the Configuration and stores all information in the Attributes Database. The Attributes Database stores the information extracted from the USB device. It maintains a pre-defined set of complete and legitimate keyboard device information and updates and supplements new device information as soon as it is identified. When simulating a device, it prioritizes the use of previously extracted device information if available; otherwise, it utilizes predefined keyboard device information.

C. Emulation and Attack Execution

When the connected device to HERMES is a host, the Sniffing Daemon calls BuildUSB to execute USB emulation. BuildUSB requests the Attributes Database to construct the necessary attributes for USB, covering all descriptors of USB as illustrated in the Fig. 5. Finally, it configures the USB Gadget device through the configfs in Linux, which is a kernel object manager based on the file system. Through configfs, users can configure the usb_gadget mode supported by the libcomposite module in user space. After generating the device directory, BuildUSB sequentially writes in Descriptor attribute values and creates a directory, /strings, to store detected language descriptor strings. Then, it links each Interface Descriptor to the corresponding Configuration Descriptor, and registers this device to the system's UDC

driver, thereby successfully emulating HERMES as a USB device. Once the device is established, BuildUSB calls the attack module after the simulated device is recognized by the host, executing pre-stored attack scripts. HID Attack is tasked with interfacing with the device file of the simulated USB device. It accomplishes this by writing HID messages to the file. The driver for the virtual USB device, such as g_hid.ko, actively monitors write operations to the virtual USB device file and forwards the written data to the USB core. Subsequently, after the USB core encapsulates the data into USB packets, the UDC driver facilitates the direct transmission of these packets through the USB bus to the connected target host, thereby facilitating HID attacks. To enable HERMES to achieve Plug and Play functionality, once the simulated USB device is recognized by the target host, HIDAttack requests attack payloads from the Payload Storage and then writes them into the device file to initiate HID attacks on the target host.

HERMES also provides a platform for attackers to craft more intricate and sophisticated attack payloads. The Remote Control feature in HERMES activates when HERMES is powered on. It utilizes a WiFi module to transform HERMES into a WiFi Access Point (AP). When assailants connect to the BSS (Basic Service Set) network created by HERMES, they can upload their attack scripts. These scripts are then stored in the Payload Storage. Upon connection to the target host, HERMES transmits these customized scripts instead of the preloaded ones stored internally. Moreover, Remote Control offers real-time control services. When employing HERMES remotely, entering keyboard commands leads to the mapping of these commands to corresponding HID reports. Subsequently, these reports are written into the simulated USB device file, facilitating the execution of corresponding operations on the target host.

V. Evaluation

This section details the implementation environment of HERMES, the results of its implementation, the penetration capabilities of various firewalls, and comparisons with various attack simulation platforms.

A. Implementation Environment

Regarding hardware, HERMES utilizes the Raspberry Pi Zero 2 W, selected for its affordability, compact size, and portability, enhancing the agility and versatility of HERMES as a USB attacking tool. Equipped with an OTG micro USB port and the DWC2 (DesignWare USB 2.0 Controller) as the USB device controller, the Raspberry Pi Zero 2 W serves as the foundation for HERMES operations. Additionally, HERMES's remote functionality is based on the WiFi module of this development board.

For the system aspect, we employ Raspbian, a Debian-based operating system, with an armv7l architecture (kernel version 6.1.21-v7+). We utilize the kernel's libcomposite module to enable USB Gadget mode and construct USB Gadget devices using configfs, a kernel object manager based on the file system. Through configfs, we configure the usb_gadget mode supported by the libcomposite module in user space.

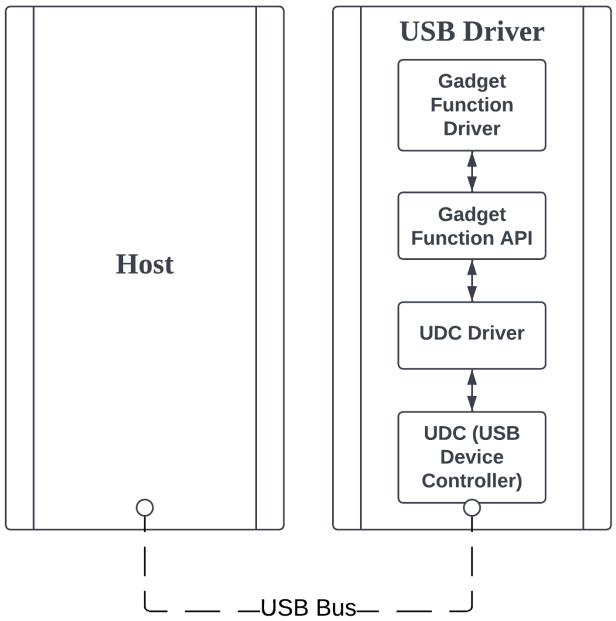

Fig. 7. The component architecture of device driver HERMES would emulate.

TABLE I
Implementation results of identification on Windows.

Feature	Logitech Wireless Mouse Receiver	HERMES
Driver	\SystemRoot\System32\drivers\usbccgp.sys	
Class GUID	{36fc9e60-c465-11cf-8056-444553540000}(GUID_DEVCLASS_USB)	
Status	0x0180600A	
Device Description	USB Composite	
USB Version	2.0	
Demanded Current	98mA	

B. Implementation Results

In our experiments, we initially inserted Logitech's wireless mouse USB receiver into HERMES to allow HERMES to clone this device. Subsequently, we inserted both the mouse receiver and HERMES into a Windows host. The Table I below presents the identification results on Windows:

We observed that Windows classified HERMES and its cloned device into the same class, loading the same drivers for both devices. This allows HERMES to possess identical driver characteristics as the cloned device, including Driver Description, Driver Version, and Driver Letter, enabling HERMES to bypass firewall identification based on these criteria.

Table II shows the Device Descriptor identification results, HERMES successfully received and replicated all of the Device Descriptor attributes. However, due to limitations of the UDC, DWC2, system-on-chip (SoC) design on the Raspberry Pi Zero 2 W, when simulating a bMaxPacketSize0 less than 64, it is limited to 64. Fortunately, we did not discover any identification targeting bMaxPacketSize0 in current USB

TABLE II
THE IDENTIFICATION RESULTS OF DEVICE DESCRIPTOR.

Feature	Logitech Wireless Mouse Receiver	HERMES
idVendor	0x046D (Logitech Inc.)	
idProduct	0xC53F	
bDeviceClass/ SubClass/Protocol	00/00/00	
iManufacturer	Logitech	
iProduct	USB Receiver	
bcdUSB	200	
bMaxPacketSIze0	32	
bcdDevice	4401	
iSerialNumber	⟨NULL⟩	
bNumConfigurations	01	

TABLE III
THE IDENTIFICATION RESULTS FOR THE CONFIGURATION DESCRIPTOR.

Feature	Logitech Wireless Mouse Receiver	HERMES
bNumInterfaces	3	
bConfigurationValue	1	
iConfiguration	RQR44.01_B0005	
bmAttributes	0xA0	
MaxPower	98mA	

firewall systems. Therefore, in our subsequent tests, HERMES continues to bypass contemporary USB firewalls.

Table III illustrates the identification results for the Configuration Descriptor. HERMES accurately replicated all Configuration Attributes.

Since the Logitech Wireless Mouse Receiver comes with a pre-existing Keyboard Interface, HERMES does not need to create a new one. It directly copies the HID report descriptor. Table IV shows the identification results for the Keyboard Interface Descriptor. HERMES successfully emulates and is capable of sending HID commands based on this Interface.

TABLE IV
THE IDENTIFICATION RESULTS FOR THE KEYBOARD INTERFACE DESCRIPTOR.

Feature	Logitech Wireles Mouse Receiver	HERMES
bInterfaceNumber	0	
bNumEndpoints	1	
bInterfaceClass/ SubClass/Protocol	03/01/01	
iInterface	⟨NULL⟩	
report_descriptor	//x05//x01//x09//x06//xa1//x01//x05//x07 x19//xe0//x29//xe7//x15//x00//x25//x01// x75//x01//x95//x08//x81//x02//x81//x03// x95//x05//x05//x08//x19//x01//x29//x05// x91//x02//x95//x01//x75//x03//x91//x01// x95//x0a//x75//x08//x15//x00//x26//xa4// x00//x05//x07//x19//x00//x2a//xa4//x00// x81//x00//xc0	

TABLE V
MOST OF THE KEY FEATURES USED BY USBGUARD [7], SOLARWINDS USB DEFENDER [21], SYMANTEC DATA LOSS PREVENTION (DLP) [9], IVANTI ENDPOINT SECURITY [11], ETC [1], [8], [10], [12], [13], [20]., FIREWALLS FOR DEVICE IDENTIFICATION.

Hardware ID	Vendor ID	bmAttributes
Serial Number	Product ID	Configuration Info
Driver Version	Driver Info	Class/SubClass/Protocol
Power	Manufacturer Info	Config Value
Device Release Number	Device Name	Class GUID

C. Penetration Performence

To evaluate the penetration performance of HERMES against contemporary USB firewalls, we conducted a comprehensive survey of various firewall solutions. Our findings aligned with the observations presented in these works [16], [27], further validating the vulnerabilities inherent in relying solely on static device properties for authentication.

Through our survey, we identified several notable USB firewalls that employed different techniques for fingerprinting and allowlisting devices. Table V summarizes the key features used by these firewalls for device identification.

Based on our analysis, we hypothesized that HERMES, with its ability to dynamically replicate all device descriptors, would be capable of bypassing these firewalls and executing unauthorized HID attacks on protected systems.

To validate our hypothesis, we selected a subset of the surveyed firewalls for practical evaluation, including USB Filter, USB Guardian, USB Guard, and the Windows Server Firewall. Our tests involved attempting to infiltrate systems protected by these firewalls while impersonating allowlisted devices.

The results of our evaluation were positive, as HERMES successfully bypassed all tested firewalls and executed malicious HID payloads on the target systems undetected. These findings further underscore the vulnerabilities in relying solely on static device properties for authentication, as HERMES can dynamically adapt and mimic the characteristics of allowlisted devices in real-time.

The Table VI presents a comparison of various platforms used for simulating USB devices, including HERMES, Psychon [30], Rubber Ducky [5], and P4wnP1 [29]. Each platform is evaluated across several criteria, highlighting their respective capabilities and limitations. In terms of hardware cost, HERMES and P4wnP1 fall into the middle range, while Psychon is considered a low-cost option, and Rubber Ducky is relatively expensive. One key advantage of HERMES and P4wnP1 is that they do not require re-flashing firmware, making them more convenient to use compared to Psychon and Rubber Ducky, which necessitate firmware updates.

HERMES stands out as the only platform capable of dynamically simulating USB devices, a feature not available in the other platforms evaluated. Regarding the ability to pass firewalls, HERMES is the sole platform that can successfully bypass firewall defenses, while the others are susceptible to detection.

When it comes to simulating various USB functions, HERMES boasts the ability to mimic every USB type, surpassing

TABLE VI
COMPARISON OF VARIOUS PLATFORMS, INCLUDING HERMES, PSYCHON, RUBBER DUCKY, AND P4WNP1, USED FOR SIMULATING USB DEVICES.

	HERMES	**Psychon[10]**	**Rubber Ducky[5]**
Hardware cost	middle	low	high
Re-flash firmware	no	yes	yes
Dynamic simulate	yes	no	no
Pass firewall	yes	no	no
Simulate functions	every USB type	HID, Storage	HID
Custom Payload Platform	yes	no	yes

the capabilities of the other platforms. Psychon and Rubber Ducky are limited to simulating HID and Storage devices, while P4wnP1 can simulate HID, Storage, RNDIS, and CDC ECM. Finally, HERMES and Rubber Ducky offer the flexibility of custom payload platforms, allowing users to create and execute tailored payloads, a feature not available in Psychon and P4wnP1.

In summary, HERMES emerges as a versatile and powerful platform, offering dynamic simulation capabilities, the ability to bypass firewalls, support for simulating any USB type, and a custom payload platform, making it a formidable tool in the USB attack landscape.

D. Comparision

VI. CONCLUSION

Our research presents HERMES as an innovative and lightweight USB attack method capable of replicating plug-and-play device characteristics, thereby enabling it to intrude and bypass the and majority of conventional USB firewall defenses in real-time. Also, we designed HERMES providing a platform remotely for adversaries to craft more intricate and sophisticated attack payloads and transmit customized scripts to the target host in real-world scenarios. Moreover, We tested HERMES in advanced security environments, where only pre-registered input devices such as mice and keyboards are permitted for USB connectivity, HERMES could still sniff the descriptor packet and bypass the USB firewall and existing defense mechanism to execute the exploit script.

By demonstrating the evaluation of HERMES against various USB defending methods, our study reaffirms the persistent threat of BadUSB, a distinct vulnerability inherent within contemporary USB and Human Interface Device (HID) systems. Furthermore, beyond raising awareness of the potential risks associated with USB-based attacks, we look forward to suggest that researchers in the field of USB or HID security can serve HERMES as a novel approach for testing and validating the effectiveness of various defenses and mitigation methods, contributing to the ongoing efforts against mainly in emerging BadUSB vulnerabilities and USB-based threats.

REFERENCES

[1] D. J. Tian, N. Scaife, A. Bates, K. Butler, and P. Traynor, "Making USB Great Again with USBFILTER," in *25th USENIX Security Symposium (USENIX Security 16)*, USENIX Association, USA, August 2016, pp. 415-430.

[2] N. Nissim, R. Yahalom and Y. Elovici, "USB-based attacks," *Computers & Security*, vol. 70, pp. 675-688, September 2017. DOI: 10.1016/J.COSE.2017.08.002.

[3] D. V. Pham, A. Syed and M. N. Halgamuge, "Universal serial bus based software attacks and protection solutions," *Digital Investigation*, vol. 7, no. 3, pp. 172-184, April. 2011. DOI: 10.1016/J.DIIN.2011.02.00.

[4] K. Nohl, S. Krißler and J. Lell, "BadUSB - On Accessories that Turn Evil," in *Blackhat USA*, Aug. 2014. [Online]. Available: https://web.archive.org/web/20140808142747/https:/srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf

[5] Hak5 LLC. "USB Rubber Ducky" Hak5.org. Accessed: Feb. 15, 2024. [Online.] Available: https://shop.hak5.org/products/usb-rubber-ducky

[6] P. J. Stoffregen and R. C. Coon. "Teensy® 4.0 Development Board" PJRC.com. Accessed: Feb. 15, 2024. [Online.] Available: https://www.pjrc.com/store/teensy40.html

[7] Red Hat, Inc. "USBGuard" GitHub.com. Accessed: Mar. 19, 2024. [Online.] Available: https://github.com/USBGuard/usbguard

[8] GFI Software "GFI EndPointSecurity - Whitelisting and Blacklisting." GFI.com. Accessed: Jan. 28, 2022. [Online.] Available: https://web.archive.org/web/20220128122928/https://www.gfi.com/products-and-solutions/network-security-solutions/gfi-endpointsecurity

[9] Broadcom Inc. "Symantec Data Loss Prevention (DLP) - Device Control." Broadcom.com. Accessed: Mar. 19, 2024. [Online.] Available: https://www.broadcom.com/products/cybersecurity/information-protection/data-loss-prevention

[10] Trinit-Soft UG. "USB Wächter (USB Monitor, USB Guardian)" Trinit-Soft.de. Accessed: Oct. 10, 2019. [Online.] Available: https://web.archive.org/web/20191010203551/http://www.trinit-soft.de/freeware/usb-waechter

[11] Ivanti. "Ivanti EndPoint Security - Device Control" Ivanti.com. Accessed: Mar. 19, 2024. [Online.] Available: https://www.ivanti.com/products/device-control

[12] Everstrike Software. "StopUSB - Device Whitelist." Everstrike.com. Accessed: Feb 17, 2020. [Online.] Available: https://web.archive.org/web/20200217101714/http://www.everstrike.com/usbsecurity/help/introduction.htm

[13] Ubuntu Forums. "Guardian - USB Whitelisting Script" UbuntuForums.org. Accessed: Mar. 19, 2024. [Online.] Available: https://ubuntuforums.org/showthread.php?t=2158605

[14] G. Hernandez, F. Fowze, D. J. Tian, T. Yavuz, and K. R. B. Butler, "FirmUSB: Vetting USB Device Firmware using Domain Informed Symbolic Execution," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, Association for Computing Machinery, New York, NY, USA, 2017, pp. 2245–2262. DOI: 10.1145/3133956.3134050.

[15] M. Tischer et al., "Users Really Do Plug in USB Drives They Find," in *Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P)*, San Jose, CA, USA, May 2016, pp. 306-319, DOI: 10.1109/SP.2016.26.

[16] H. Mohammadmoradi and O. Gnawali, "Making Whitelisting-Based Defense Work Against BadUSB," in *Proceedings of the 2nd International Conference on Smart Digital Environment (ICSDE'18)*, Association for Computing Machinery, New York, NY, USA, October 2018, pp. 127–134. DOI: 10.1145/3289100.3289121.

[17] 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 (ACSAC '15)*, Association for Computing Machinery, New York, NY, USA, December 2015, pp. 261–270. DOI: 10.1145/2818000.2818040.

[18] D. S. Lin, Z. Y. Zeng, C. C. Hu, Y. You, (in Chinese), "USB camouflage intrusion detection method and device," CN Patent 113343240B, Mar. 1, 2024.

[19] O. V. ZaitsevOlga, E. DomkeKonstantin, Y. ManurinMikhail and A. Levinsky, "System and method for detecting modified or corrupted external devices," U.S. Patent 9 386 024 B1, Jul. 5, 2016.

[20] G Data CyberDefense AG. "USB Keyboard Guard - How to be sicher from usb attacks." GDataSoftware.com. Accessed: Mar. 19, 2024. [Online.] Available: https://www.gdatasoftware.com/en-usb-keyboard-guard

[21] SolarWinds. "Security Event Manager - USB Defender" SolarWinds.com. Accessed: Mar. 19, 2024. [Online.] Available: https://www.solarwinds.com/security-event-manager/use-cases/usb-security-analyzer

[22] Comptia. “Cyber secure: a look at employee cybersecurity habits in the workplace,” 2015, [online] Available: https://web.archive.org/web/20190930214446/https://www.comptia.org/resources/cyber-secure-a-look-at-employee-cybersecurity-habits-in-the-workplace
[23] S. Golovanov. “DarkVishnya: Banks attacked through direct connection to local network.” Kaspersky, Securelist.com. Accessed: Apr. 2, 2024. [Online.] Available: https://securelist.com/darkvishnya/89169
[24] I. Ilascu. “FBI: Hackers Sending Malicious USB Drives & Teddy Bears via USPS.” Bleepingcomputer.com. Accessed: Apr. 2, 2024. [Online.] Available: https://www.bleepingcomputer.com/news/security/fbi-hackers-sending-malicious-usb-drives-and-teddy-bears-via-usps
[25] D. J. Tian, N. Scaife, D. Kumar, M. Bailey, A. Bates and K. Butler, “SoK: ”Plug & Pray” today–understanding USB insecurity in versions 1 through C,” in *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P)*, IEEE, May 2018, pp. 1032-1047. DOI: 10.1109/SP.2018.00037.
[26] S. Angel et al., “Defending against malicious peripherals with Cinch,” in *25th USENIX Security Symposium (USENIX Security 16)*, USENIX Association, USA, August 2016, pp. 397-414.
[27] M. Al-Zarouni, “The Reality of Risks from Consented use of USB Devices,” in *Proceedings of 4th Australian Information Security Management Conference*, Edith Cowan University, Perth, Western Australia, December 2006, pp. 312-317. DOI: 10.4225/75/57b6543434762.
[28] E. L. Loe, H. C. Hsiao, T. H. J. Kim, S. C. Lee and S. M. Cheng, “SandUSB: An installation-free sandbox for USB peripherals,” in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Reston, VA, USA, December 2016, pp. 621-626, DOI: 10.1109/WF-IoT.2016.7845512.
[29] R. Dawes et al. “P4wnP1” GitHub.com. Accessed: Mar. 19, 2024. [Online.] Available: https://github.com/mame82/P4wnP1
[30] A. Caudill and B. Wilson. “Psychson” GitHub.com. Accessed: Mar. 19, 2024. [Online.] Available: https://github.com/brandonlw/Psychson
[31] Kingston Technology. “Kingston Ironkey USB Flash Drive” Kingston.com. Accessed: Feb. 15, 2024. [Online.] Available: https://www.kingston.com/en/solutions/data-security/ironkey