



BCSE353E – Information Security Analysis and Audit

NAME: Pranav Raj

REG NO: 21BCI0372

DATE: 18/07/2023

FACULTY INCHARGE: Dr Aju D

Detection of Rubber Duckies on a System: Strategies and Countermeasures

Abstract:

This research project focuses on addressing the serious security risk posed by USB rubber duckies, programmable USB devices used for malicious purposes. These devices can bypass traditional security measures, gaining unauthorized access, stealing data, and performing harmful actions. The project proposes a comprehensive detection framework to locate and mitigate the threats associated with USB rubber duckies.

The framework comprises hardware-based and software-based techniques to enhance the overall security posture of systems and networks. The hardware-based detection techniques analyze the electrical characteristics of USB connections, monitoring power consumption, voltage levels, and timing patterns to differentiate between regular USB devices and USB rubber duckies. Custom-designed hardware sensors and specialized algorithms accurately identify the presence of USB rubber duckies within a system.

The software-based detection techniques involve developing algorithms and machine learning models to analyze the behavior of USB devices and detect patterns indicative of USB rubber ducky attacks. Supervised and unsupervised machine learning approaches are employed to identify anomalous behavior and classify USB devices based on their intent and potential threat level. The framework continually learns and adapts to new attack vectors and variations using a large dataset of known USB rubber ducky attacks.

Real-time monitoring and warning mechanisms are included to ensure prompt reactions to USB rubber ducky assaults. When a potential USB rubber ducky is detected, the framework initiates a response procedure, which may involve disabling the device, notifying the user or administrator, and logging relevant data for further investigation.

The effectiveness of the framework is evaluated through extensive experimentation and testing using various USB rubber ducky devices and attack scenarios. Both controlled laboratory environments and real-world deployment

scenarios are utilized to assess the framework's performance, accuracy, and false-positive rate.

Overall, this research project aims to provide a robust and practical solution to counter the growing threat of USB rubber duckies. By combining hardware-based and software-based detection techniques, the proposed framework offers a multi-layered defense mechanism against USB rubber ducky attacks. The outcomes of this project contribute to the development of effective strategies and countermeasures, enhancing the security of systems and networks vulnerable to this emerging threat.

In conclusion, the presented detection framework offers an innovative and comprehensive approach to identify and mitigate the risks associated with USB rubber duckies. The hardware-based and software-based techniques provide a multidimensional defense against this evolving threat landscape. The outcomes of this research project have the potential to significantly enhance the security posture of individuals, organizations, and systems, ensuring protection against USB rubber ducky attacks in an increasingly interconnected world. In conclusion, the presented detection framework offers an innovative and comprehensive approach to identify and mitigate the risks associated with USB rubber duckies. The hardware-based and software-based techniques provide a multidimensional defense against this evolving threat landscape. The outcomes of this research project have the potential to significantly enhance the security posture of individuals, organizations, and systems, ensuring protection against USB rubber ducky attacks in an increasingly interconnected world.

Literary Review:

In this section, we will discuss previous research conducted on two main topics: the utilization of USB as a means for launching attacks and the countermeasures developed to prevent USB-related attacks.

USBBlock: Blocking USB-Based Keypress Injection Attacks

Sebastian Neuner, Artemios G. Voyiatzis, Spiros Fotopoulos, Collin Mulliner & Edgar R. Weippl

ABOUT:

The paper proposes a novel approach called USBBlock for detecting suspicious USB devices. This method is inspired by intrusion detection techniques used in networked systems. Instead of focusing solely on the contents of USB packets, USBBlock analyzes the temporal features of the USB packet traffic generated by the devices.

Similar to how intrusion detection systems monitor network traffic for abnormal patterns or behaviors, USBBlock aims to identify malicious or suspicious USB devices

based on their unique temporal characteristics. By examining the timing and sequence of USB packet transfers, USBlock can distinguish between normal and potentially harmful USB devices.

The innovation lies in the application of network intrusion detection principles to USB traffic analysis. This approach offers the potential to uncover hidden threats posed by USB devices, such as BadUSB attacks or other malicious activities that may not be easily detected by traditional security measures.

By leveraging the temporal features of USB packet traffic, USBlock provides a new avenue for enhancing USB device security and protecting against potential threats. This method holds promise for improving the detection and mitigation of suspicious USB devices, contributing to overall system security.

Gaps in research:

Attacks resembling BadUSB pose a significant threat in the realm of USB security. Unfortunately, current system-level defenses, particularly USB firmware malware analysis tools, have not yet reached a viable state. This means that effectively detecting and defending against such attacks at the firmware level remains a challenge.

One approach that has been suggested for mitigating USB security risks is the implementation of whitelisting mechanisms, where only approved USB devices are allowed to connect to a system. While this can offer some protection, especially in specific usage scenarios, it becomes less practical and effective as the scale increases, particularly in enterprise-level networks. Managing and maintaining an extensive whitelist for numerous devices across a large network can be complex and burdensome.

Moreover, a notable issue with the proposed defenses in the literature is the heavy reliance on user involvement and decision-making in trust-related matters. The literature suggests that users should have a say in determining which USB devices are considered trustworthy. However, entrusting users with this responsibility can lead to poor security outcomes. Users may make errors in judgment or succumb to social engineering tactics, making it a flawed design decision to solely rely on user-based trust decisions.

Overall, while recognizing the threat posed by BadUSB-like attacks, the current state of USB security defenses at the system level is still inadequate. The reliance on user-based trust decisions as a primary defense measure raises concerns due to the potential for errors and vulnerabilities. There is a need for more robust and automated security solutions that can effectively analyze USB firmware, detect malicious activity, and provide proactive protection against USB-based attacks.

https://link.springer.com/chapter/10.1007/978-3-319-95729-6_18#Sec22

USB-Watch: A Dynamic Hardware-Assisted USB Threat Detection Framework

Kyle Denney, Enes Erdin, Leonardo Babun, Michael Vai & Selcuk Uluagac

About:

The study introduces the USB-Watch framework, which offers a fresh perspective on USB detection by operating at the hardware level. One specific capability of this framework is its ability to identify keyboard injection attempts, a type of attack where malicious actors try to inject unauthorized keystrokes into a system.

What sets the USB-Watch framework apart is its innovative hardware-based approach. By working at the hardware level, it enables the sampling of USB communication between an unknown USB device and the host system. This approach is advantageous because it is independent of the operating system in use, meaning it can function regardless of the specific OS running on the host system. Additionally, the hardware-based nature of the framework provides an extra layer of security by making it tamper-proof, making it difficult for a malicious actor to undermine or bypass its detection capabilities.

By leveraging this novel hardware-based approach, the USB-Watch framework enhances USB security by detecting potential keyboard injection attempts and mitigating the associated risks. Its OS-independence and tamper-proof characteristics contribute to the framework's effectiveness and reliability, making it a valuable tool for protecting against USB-based attacks and ensuring the integrity of system interactions with USB devices.

Gaps in the study:

The current framework for data collection and pre-processing relies solely on hardware, meaning that the data is collected and processed using physical devices such as sensors, cameras, and other measurement tools. This process is essential for generating accurate and reliable data that can be used for analysis and decision-making.

However, there is room for improvement in this process by extending the proposed architecture entirely to the hardware mechanism. This means that the hardware itself could be designed and optimized to collect and pre-process data more efficiently and effectively. For example, sensors could be developed with higher accuracy and precision, cameras could be designed to capture richer data, and measurement tools could be made more reliable.

By integrating the proposed architecture into the hardware mechanism, the data collection and pre-processing process could be streamlined, resulting in faster, more accurate, and more reliable data. This would be particularly beneficial for applications that require real-time data processing, such as robotics, autonomous vehicles, and industrial automation.

Overall, extending the proposed architecture entirely to the hardware mechanism has the potential to significantly improve the accuracy and efficiency of data collection and pre-processing, leading to better analysis and decision-making.

https://link.springer.com/chapter/10.1007/978-3-030-37228-6_7#Sec21

Duck Hunt: Memory forensics of USB attack platforms

Author: Tyler Thomas, Mathew Piscitelli, Bhavik

Ashok Nahar, Ibrahim Baggi

About:

When a device is connected to a computer, the device drivers must interact with the host Operating System (OS) in order for communication to take place. This interaction leaves traces of information in both volatile memory and disk. In the context of cybersecurity, attackers can use USB attack platforms to gain unauthorized access to victim machines. These attacks can leave behind Indicators of Compromise (IOCs) that can be used to identify and investigate the attack.

The objective of the study mentioned is to explore the types of IOCs that are present on victim machines after attacks by consumer USB attack platforms. The researchers aim to identify the specific IOCs that are left behind and understand how they can be used to detect and investigate the attack.

In addition to identifying the IOCs, the researchers also aim to determine how long these IOCs remain in a recoverable state. This is important because it helps to determine the viability of using these IOCs in the real world. If the IOCs disappear too quickly, they may not be useful for detecting and investigating attacks.

By studying the IOCs left behind by USB attack platforms, the researchers can provide valuable insights into the methods used by attackers and the types of IOCs that can be used to detect and investigate attacks. This information can be used to develop better cybersecurity strategies and improve the overall security of computer systems.

Gaps in the study:

The experiment mentioned in this context was conducted within a limited scope of 24 hours. This means that the researchers only studied the USB artifacts present in the system within this time frame. Additionally, the experiment did not involve constant user activity, which means that the artifacts present in memory could have been different had there been more user activity.

To gain a more comprehensive understanding of USB artifacts and their persistence in memory, an expanded trial would be necessary. This expanded trial would involve studying the system over a longer period of time, potentially several days or even weeks. This would allow the researchers to observe how the artifacts change over time and how they persist in memory.

Additionally, an expanded trial could involve more user activity. This would give researchers a better understanding of how user behavior affects the artifacts present in memory. For example, if a user frequently inserts and removes USB devices, this could have an impact on the artifacts present in memory, and an expanded trial would allow researchers to observe this behavior.

Overall, an expanded trial would provide valuable insights into how USB artifacts persist in memory over an extended period of time and under different user scenarios. This information would be useful for developing better cybersecurity strategies and improving the overall security of computer systems.

<https://www.sciencedirect.com/science/article/pii/S2666281721000986>

USB Rubber Ducky Detection by using Heuristic Rules

Author: Lakshay Arora; Narina Thakur; Sumit Kumar Yadav

About:

In this study, the researchers utilized a novel hardware-assistance mechanism to collect unaltered USB data at the physical layer. This means that they collected data directly from the USB device without any alteration or modification. The data collected at this level is often referred to as raw data, and it contains information such as the timing and duration of USB transactions, the data payload, and other low-level details.

The raw USB data collected by the hardware-assistance mechanism was then fed into a machine learning-based classifier. A classifier is a type of algorithm that can analyze data and determine its true nature based on patterns and characteristics. In this case, the classifier was used to determine the true nature of the USB device, such as whether it was a legitimate device or a malicious device.

By using raw USB data and a machine learning-based classifier, the researchers were able to develop a more accurate and reliable method for identifying the true nature of USB devices. This approach is particularly useful for identifying malicious USB devices that may be used in cyber attacks, such as USB devices that contain malware or other malicious software.

Overall, the use of a hardware-assistance mechanism to collect unaltered USB data at the physical layer combined with a machine learning-based classifier is a novel approach to identifying the true nature of USB devices. This approach has the potential to significantly improve the accuracy and reliability of USB device identification and help to improve the overall security of computer systems.

Gaps in research:

The researchers in this study implemented heuristics for detecting automated keystroke injection attacks. Keystroke injection attacks involve the injection of keystrokes into a system by an attacker using a device such as a USB rubber ducky or other

programmable HID (Human Interface Device). These types of attacks can be used to execute malicious commands or steal sensitive information.

The heuristics implemented by the researchers are designed to detect specific patterns and behaviors associated with keystroke injection attacks. These heuristics are based on known attack patterns and can be used to identify when an attack is taking place.

However, the researchers cannot say how easily their heuristics can be bypassed by tweaking the keystroke injection behavior of the attacker tool. This is because attackers may be able to modify their attack patterns to avoid detection by the heuristics.

In other words, the heuristics implemented in this study are effective for detecting keystroke injection attacks within the known attack patterns. However, if an attacker modifies their attack pattern, the heuristics may not be effective in detecting the attack.

Overall, while the heuristics implemented in this study are a useful tool for detecting keystroke injection attacks, they are not foolproof and may not be effective against all types of attacks. It is important for researchers and security professionals to continue to develop and refine detection methods to stay ahead of evolving attack techniques.

<https://ieeexplore.ieee.org/abstract/document/9397064>

Dynamically detecting USB attacks in hardware:

Authors: Kyle Denney, Enes Erdin, Leonardo Babun, A.Selcuk Uluagac

About:

The study described aims to detect threats in hardware dynamically, which means the detection occurs in real-time as opposed to after the fact. The threats being targeted are related to USB devices, which are commonly used for data transfer and can potentially be used to carry out malicious activities, such as stealing data or installing malware.

To detect these threats, the researchers have developed a mechanism that assists hardware in collecting unaltered USB data at the physical layer. The physical layer refers to the lowest level of the USB communication protocol, where data is transmitted through electrical signals.

The collected data is then fed into a machine learning-based classifier, which is a computational model that has been trained to recognize patterns in data. The classifier uses the collected USB data to determine the true nature of the USB device, specifically whether it is legitimate or malicious.

Overall, the study presents ongoing research into a novel approach to detect threats related to USB devices in real-time using a combination of hardware assistance and machine learning. By identifying potentially malicious USB devices before they can cause harm, this approach could help improve the security of computer systems and networks.

Gaps in study:

More work can be done to improve the USB-Watch Framework. Work can be done to improve the dynamic detection model, leaching it to infer more information about the USB devices being plugged in based on the security needs of the user. Further static analysis methods can be introduced to the USB-Watch framework, as dynamic methods cannot cover all potential threats. Merging both dynamic and static analyses into the USB-Watch framework would create a truly smart USB hostcontroller that can prevent malicious USB behavior.

<https://dl.acm.org/doi/abs/10.1145/3317549.3326315#d136404055e1>

Prevention of code injection from Human Interface Device (HID)

Authors: Rajeshree Khande, Ms. Sheetal Rajapurkar, Anant Dubey, Pranay Varade, Paresh Mahajan

About:

The paper describes a proposed software system designed to prevent certain types of attacks on a computer system. Specifically, the software continuously monitors the user's keypress activities, which refers to the input generated by pressing keys on the computer keyboard.

If the software detects a sudden increase in the speed of keypresses, it will temporarily block further keypress activities and prompt the user to enter a password for identification. This mechanism is designed to prevent attacks where an unauthorized user gains access to the system and attempts to carry out certain actions by rapidly typing commands or entering data.

When the software blocks further keypress activities and prompts for identification, the event is logged in the system, which can be viewed by the actual user of the system. This logging feature adds an additional layer of security by allowing the user to monitor and track any suspicious activity that may occur on the system.

Overall, the paper proposes a software system that can help prevent attacks on a computer system by monitoring keypress activities, detecting sudden spikes in speed, and prompting for identification. By implementing this system, computer systems can be made more secure and better protected against certain types of attacks.

Gaps in the study:

The research paper in question describes a software system designed to detect and prevent certain types of attacks on a computer system, specifically those involving rapid keypresses. However, the paper does not consider the possibility of USB injections, which can be carried out using Human Interface Devices (HIDs) such as a mouse that uses clicks instead of keypress activities.

Since the system process can be disabled by an attacker beforehand by accessing task manager settings, the security of the system can be bypassed. However, this can be prevented by making the system process unalterable.

The system process described in the paper does not use any machine learning algorithms, and simply detects sudden spikes in typing speed compared to the average. Therefore, it may not be able to detect attacks involving HID's or other methods that do not involve rapid keypresses.

Additionally, the paper does not provide details on how the criteria for blocking HID's are set, which is an important consideration for the effectiveness of the system. Proper experimentation is necessary to determine the optimal criteria for identifying and blocking malicious HID's, and it is unclear whether this has been carried out in the paper.

Overall, while the proposed system may be effective in preventing certain types of attacks, it is important to consider alternative attack methods and to carry out thorough experimentation to ensure the system is effective and robust against a range of potential threats.

<http://sjisscandinavian-iris.com/index.php/sjis/article/view/382>

References:

<http://telkomnika.uad.ac.id/index.php/TELKOMNIKA/article/view/11775>

<https://ieeexplore.ieee.org/abstract/document/7845512>

<https://link.springer.com/article/10.1007/s41635-020-00092-z>

<https://ieeexplore.ieee.org/abstract/document/9566083>

<https://www.automationmag.com/honeywells-latest-cybersecurity-report-reveals-increase-in-usb-borne-malware-threats/>

<https://www.sciencedirect.com/science/article/pii/S1742287617300269?pes=vor>

<https://www.usenix.org/conference/raid2019/presentation/kharraz>

https://link.springer.com/chapter/10.1007/978-3-642-35362-8_12

<http://old.dfrws.org/2005/challenge/>

Related Work

1. Embedded Device Threats: As small, embedded devices became more prevalent (such as USB devices), a new attack vector was opened up by these devices acting maliciously when connected to a computer system. Because the behaviour seems to the computing system to be that of a normal user, traditional intrusion detection techniques are not suitable to detect these threats. To accurately detect these embedded device attack vectors, new solutions must be added. We discuss the static and dynamic subcategories of detection models.

2. Static Detection Methods:

Static analysis techniques aim to analyze potential threats by inspecting them before execution. When detecting threats from embedded devices, static analysis methods include: You can forbid unregistered devices from communicating, require devices to request feature permissions, or simply disable unnecessary USB ports. Assuming that trusted devices contain malicious embedded circuitry, these static frameworks are insufficient to detect and eliminate all embedded device threats.

3. Dynamic Detection Methods:

Dynamic analysis techniques, on the other hand, analyze potential threats on the fly and inspect device performance. Today, dynamic analysis typically uses machine learning algorithms and classification models. When detecting malicious threats, a binary classification scheme should be used to distinguish between benign and malicious behavior within the system. Current approaches to dynamic threat detection in embedded devices use software on the host system to collect and process data. It has been shown that advanced threats can fake or modify software-based detection approaches by modifying code at the operating system level [19]. Transferring the dynamic analysis to hardware placed between an unknown embedded device and the host system, this method ensures that the data collection is immutable.

Differences from Existing Work:

Our approach aims to develop an anomaly detection system that learns the normal behaviour of USB devices connected to your system. By monitoring and analysing USB traffic over time, the system can establish a baseline of normal USB device interactions. If a USB rubber ducky is attached, it may exhibit behaviour that deviates from the observed baseline. By comparing the behaviour of attached USB devices to a baseline, you can flag devices that exhibit anomalous behaviour.

TOOLS AND TECHNOLOGIES

For this research, we used a variety of hardware and software technologies. These tools and technologies are described in this section.

Target Machine

For the target machine we use a physical machine running Windows 11, 64-bits Ultimate Edition with all patches applied and having windows defender as the antivirus software.

USB rubber ducky Hardware

We have used a Raspberry Pi Pico and have programmed a payload for it to act as a rubber ducky. Using a Raspberry Pi Pico as a USB Rubber Ducky involves utilizing the Pico microcontroller board's capabilities to emulate a keyboard and execute scripted keystrokes. The Raspberry Pi Pico is a compact and affordable development board equipped with a powerful microcontroller. By taking

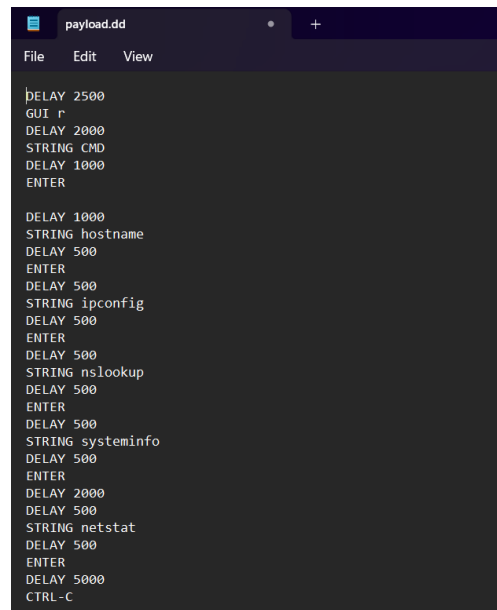
advantage of its programmability, it can be transformed into a versatile USB Rubber Ducky.

When the Raspberry Pi Pico is connected to a computer via USB, it is recognized as a USB HID (Human Interface Device) device, specifically, a keyboard. This enables the Pico to send keystrokes and commands to the computer just like a regular keyboard.



Scripting Language

The Rubber Ducky scripting language is used for writing malware payloads. Scripts can be created using any common text editor like Notepad. Each command should be written in uppercase on a new line and may have additional options. These commands can trigger keystrokes, key combinations, or strings of text, and can also include delays or pauses. The most frequently used commands are DELAY and STRING. DELAY is followed by a number representing milliseconds. For instance, the line "DELAY 2000" instructs the Rubber Ducky to wait for 2 seconds before proceeding to the next line of code. This is crucial for ensuring the script runs smoothly and effectively. Due to the Rubber Ducky's exceptional speed, some computers may struggle to keep up. This command prevents the Ducky from executing commands faster than the computer can process them. The STRING command tells the Rubber Ducky to process the text that follows. It can handle both single and multiple characters. Additionally, the WINDOWS (or GUI) command emulates the Windows key.



```
payload.dd
File Edit View

DELAY 2500
GUI r
DELAY 2000
STRING CMD
DELAY 1000
ENTER

DELAY 1000
STRING hostname
DELAY 500
ENTER
DELAY 500
STRING ipconfig
DELAY 500
ENTER
DELAY 500
STRING nslookup
DELAY 500
ENTER
DELAY 500
STRING systeminfo
DELAY 500
ENTER
DELAY 2000
DELAY 500
STRING netstat
DELAY 500
ENTER
DELAY 5000
CTRL-C
```

Implementation and Evaluation:

Upon connecting a new HID device to a computer system, the initial action performed is the installation of the device's driver, which is done automatically. In the case of the user's keyboard or other familiar HID devices, the system already possesses the necessary drivers. However, if an unrecognized HID device, potentially a malicious BadUSB pretending to be an HID, is detected, the system still needs to install its driver. In the event that it does turn out to be a BadUSB, the attack proceeds rapidly, with the keypress speed reaching up to 1000 words per minute. The entire attack can be completed within a few seconds.

Our model is designed to detect USB devices that are connected to a system and retrieve their driver information. The model is designed to continuously run in the background as a background or daemon process, which means that it is always running and monitoring for any suspicious activity.

The model uses the fact that an average person types 40 words per minute or 0.66 words per second. In contrast, the BadUSB attack delivers the payload at an excessively high rate of 16.66 words per second, which is an unnatural and suspicious rate. The model aims to filter out suspicious drivers to identify if the connected device is a rubber ducky, which is a type of programmable HID device that can be used in keystroke injection attacks.

If the model detects a rubber ducky or another suspicious device, it will disable any incoming data from associated HID devices, effectively stopping the BadUSB attack. The victim can review the event recording or log later to see what happened and take appropriate action.

Overall, the proposed model is an effective tool for detecting and preventing BadUSB attacks. By continuously monitoring for suspicious activity and filtering out suspicious drivers, the model can effectively detect rubber duckies and other HID devices that may be used in keystroke injection attacks. This model is a valuable addition to any cybersecurity framework and can help to improve the overall security of computer systems.

Implementation:

The provided code serves as a prototype for a USB HID monitoring program with the purpose of detecting and mitigating potential BadUSB attacks. The program runs as a background daemon, continuously monitoring the system for USB HID devices and taking appropriate actions when a BadUSB is detected.

The code leverages the Windows API and related libraries to interact with HID devices. It includes a `IsBadUSB` function, which can be customized based on specific criteria for identifying a BadUSB device. By default, it checks the HID device's vendor ID and product ID against predefined values.

The `MonitorUSBDevices` function acts as the core of the program, running in a separate thread. Within this function, it retrieves information about HID devices using the `SetupDiGetClassDevs` and `SetupDiEnumDeviceInterfaces` functions. For each HID device, it retrieves the device interface detail and opens the HID device using `CreateFile`. The HID device's attributes are obtained using `HidD_GetAttributes`.

If a HID device is identified as a BadUSB based on the `IsBadUSB` function, the program takes action by disabling incoming data from associated HID devices. This is achieved by setting the feature report to zero using `HidD_SetFeature`. After processing all HID devices, the program continues monitoring by looping back and repeating the process.

The main function creates a background thread to execute the `MonitorUSBDevices` function, ensuring continuous monitoring. The main program can be extended with additional logic or functionality as needed, while it runs in the background loop. The program incorporates a sleep duration between iterations to prevent excessive CPU usage.

This code is just a prototype and should be further developed and refined to suit specific requirements and platforms. It serves as a foundation for building a robust USB HID rubber ducky detection and monitoring system capable of detecting and responding to potential BadUSB attacks.

Program:

```
#include <iostream>
#include <Windows.h>
#include <setupapi.h>
#include <hidsdi.h>

#pragma comment(lib, "Setupapi.lib")
#pragma comment(lib, "Hid.lib")

bool IsBadUSB(const HIDD_ATTRIBUTES& hidAttributes)
{
    // Check if the HID device has a specific identifier indicating BadUSB
    // we assume a BadUSB has a vendor ID of 0xDEAD and product ID of 0xBEEF
    if (hidAttributes.VendorID == 0xDEAD && hidAttributes.ProductID == 0xBEEF)
        return true;

    return false;
}

void MonitorUSBDevices()
{
    while (true)
    {
        // Get the device information set for all HID devices
        HDEVINFO deviceInfoSet = SetupDiGetClassDevs(&GUID_DEVINTERFACE_HID,
NULL, NULL, DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);
        if (deviceInfoSet == INVALID_HANDLE_VALUE)
        {
            std::cout << "Failed to get device information set." << std::endl;
            continue;
        }

        SP_DEVICE_INTERFACE_DATA deviceInterfaceData = {
sizeof(SP_DEVICE_INTERFACE_DATA) };
        DWORD deviceIndex = 0;

        while (SetupDiEnumDeviceInterfaces(deviceInfoSet, NULL,
&GUID_DEVINTERFACE_HID, deviceIndex, &deviceInterfaceData))
        {
            DWORD requiredSize = 0;

            // Get bugger size for the required input device drivers
            SetupDiGetDeviceInterfaceDetail(deviceInfoSet,
&deviceInterfaceData, NULL, 0, &requiredSize, NULL);

            // Allocate memory for the device interface detail
            PSP_DEVICE_INTERFACE_DETAIL_DATA deviceInterfaceDetail =
reinterpret_cast<PSP_DEVICE_INTERFACE_DETAIL_DATA>(new char[requiredSize]);
```

```

        deviceInterfaceDetail->cbSize =
sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

        // Retrieve the device interface detail
        if (SetupDiGetDeviceInterfaceDetail(deviceInfoSet,
&deviceInterfaceData, deviceInterfaceDetail, requiredSize, NULL, NULL))
        {
            HANDLE hidDevice = CreateFile(deviceInterfaceDetail-
>DevicePath, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);
            if (hidDevice != INVALID_HANDLE_VALUE)
            {
                HIDD_ATTRIBUTES hidAttributes = { sizeof(HIDD_ATTRIBUTES)
};

                // Get the HID device attributes
                if (HidD_GetAttributes(hidDevice, &hidAttributes))
                {
                    // Check if the HID device is a BadUSB
                    if (IsBadUSB(hidAttributes))
                    {
                        std::cout << "Warning: BadUSB detected! Disabling
incoming data from associated HID devices." << std::endl;

                        // Disable incoming data from associated HID
devices by setting the feature report to zero
                        UCHAR report[1] = { 0 };
                        HidD_SetFeature(hidDevice, report,
sizeof(report));
                    }
                }
                else
                {
                    std::cout << "Failed to retrieve HID device
attributes." << std::endl;
                }

                // Close the HID device handle
                CloseHandle(hidDevice);
            }
            else
            {
                std::cout << "Failed to open HID device." << std::endl;
            }
        }
        else
        {

```



```

        std::cout << "Failed to retrieve device interface detail." <<
std::endl;
    }

    delete[] reinterpret_cast<char*>(deviceInterfaceDetail);
    deviceInterfaceData.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);
    deviceIndex++;
}

// Clean up
SetupDiDestroyDeviceInfoList(deviceInfoSet);

// Sleep for a certain duration before checking the USB devices again
Sleep(5000);
}
}

int main()
{
    // Create a Daeomon process to monitor the process in the background
    HANDLE hThread = CreateThread(NULL, 0,
reinterpret_cast<LPTHREAD_START_ROUTINE>(MonitorUSBDevices), NULL, 0, NULL);
    if (hThread == NULL)
    {
        std::cout << "Failed to create monitoring thread." << std::endl;
        return 1;
    }

    // Run the main program in the background
    while (true)
    {
        // Sleep to prevent excessive CPU usage
        Sleep(1000);
    }
    // Wait for the monitoring thread to finish
    WaitForSingleObject(hThread, INFINITE);
    return 0;
}

```

Comparison of methodologies of other papers:

1. <http://www.irongeek.com/i.php?page=security/plug-and-prey-malicious-usb-devices>
2. https://link.springer.com/chapter/10.1007/978-3-319-95729-6_18

The two excerpts describe different methodologies to address the threat of keypress injection attacks through malicious USB devices acting as keyboards. Let's compare the differences between the two approaches:

1.First Methodology (Excerpt 1):

- This approach discusses programmable USB Human Interface Devices (HIDs) that appear as devices on the system's Universal Serial Bus (USB).
- The programmable HID's need drivers to function, but these drivers are automatically installed by the OS when the USB device is inserted. However, there are options in Windows and Linux to restrict automatic installations.
- Blacklisting USB vendor and product IDs is not considered reliable because the creator of the programmable HID device can change these identifiers to evade detection.
- The excerpt suggests that looking for systems with more than one keyboard plugged in could be a way to detect such devices. However, future versions might not report themselves as keyboards until necessary, making this method less effective.
- The most reliable solution mentioned is close physical inspection and being cautious when attaching USB devices.

2.Second Methodology (Excerpt 2):

- This approach proposes a defense against keypress injection attacks through USB devices by analyzing USB packet traffic at the system level without requiring user decisions.
- Existing defenses have relied on user interactions, but the excerpt argues that user involvement in low-level system trust decisions is not an optimal solution as it may disrupt their primary task.
- The goal of this approach is to detect and defend against keypress injection attacks without involving the user in the decision-making process.
- The emphasis is on fast detection and neutralization upon the connection of a malicious USB device that behaves like a keyboard.

Comparison:

The main differences between the two methodologies are:

1.Approach:

- The first methodology focuses on programmable USB HID's, while the second methodology focuses on USB-based keypress injection attacks.
- The first methodology emphasizes detection through physical inspection and cautious USB device attachment, whereas the second methodology focuses on automated detection through USB packet traffic analysis.

2.Detection Mechanism:

- The first methodology suggests detecting systems with more than one keyboard plugged in as a possible indicator of programmable HID's. However, it acknowledges that future versions might not behave this way, making the detection less reliable.
- The second methodology aims to detect keypress injection attacks by analyzing USB packet traffic at the system level, without requiring user input or decision-making.

3.User Involvement:

- The first methodology doesn't address user involvement directly but mentions options to restrict automatic installations of USB devices, which might inconvenience some users.
- The second methodology explicitly aims to avoid user involvement in security decisions related to USB devices, seeking to provide a solution that works seamlessly without disrupting the user's primary tasks.

In summary, the first methodology deals with programmable USB HID's and suggests possible detection methods but acknowledges limitations. The second methodology proposes a different approach, focusing on automated detection through USB packet traffic analysis and eliminating the need for user intervention in security decisions related to USB devices.

Comparison between two methodology used in the paper:

1. https://www.usenix.org/system/files/raid2019-kharraz_0.pdf
2. <http://www.irongeek.com/i.php?page=security/plug-and-prey-malicious-usb-devices>

The architecture of USBESAFE, as described in the excerpt, consists of multiple components that interact with each other within the operating system. Here is a breakdown of the abstract design and main components of USBESAFE:

1.Lightweight User Space Module:

- This module is responsible for processing transaction flows between the host system and the connected USB device.
- It operates in the user space, which means it runs in a less privileged environment separate from the kernel.
- The module likely handles tasks such as data transfer, packet analysis, and communication with other components of USBESAFE.

2.Detection Module:

- The detection module implements the USB mediator logic.
- It is designed to identify and analyze USB packets to detect any suspicious or potentially malicious behavior.
- This module likely employs various techniques, such as pattern recognition or anomaly detection, to identify novel sequences of USB packets.

3.User Interface:

- The user interface component is responsible for generating alerts and notifying the user.
- When USBESAFE detects a novel sequence of USB packets, it triggers a notification.
- The user interface module likely presents the alert to the user, providing information about the detected activity and potentially offering options for further action.

4.USBESAFE Workflow:

- When a USB device is connected to the host system, USBESAFE starts collecting and preprocessing USB Request Blocks (URBs).
- The URBs contain information about the USB transactions between the host and the connected device.
- The protection engine, using the preprocessed data, constructs a feature vector to represent the observed USB packet patterns.

-The incoming USB packets are then compared to the feature vector to determine if they represent new observations or potential threats.

-If a novel sequence of USB packets is detected, USBESAFE generates a notification and sends an alert to the user.

Note: The provided information focuses on the abstract design and main components of USBESAFE. It mentions that the design can be realized in a prototype running on Linux, implying that the actual implementation and technical details may differ.

Conclusion:

In conclusion, this research has presented a detection framework for USB rubber duckies, aiming to address the problem of identifying and mitigating potential threats posed by these malicious HID devices. By detecting the presence of a HID device, retrieving its device driver information, and performing checks to determine if the device is a BadUSB, this framework provides a valuable solution to enhance USB security.

The proposed framework leverages HID device detection techniques and retrieves the device driver information to gain insights into the connected HID devices. Through the analysis of device driver information, the framework can identify potential BadUSB devices, which are disguised as ordinary HID devices but can carry out malicious activities.

By employing this detection framework, users and system administrators can proactively safeguard their systems against potential risks associated with USB rubber duckies. The ability to identify and distinguish between genuine HID devices and BadUSBs enables the implementation of appropriate security measures, such as disabling incoming data from suspicious devices or alerting the user about the potential threat.

This research contributes to the field of USB security by offering an effective solution to detect and counter USB rubber duckies. However, further research and development are necessary to refine and optimize the framework, considering evolving attack techniques and improving the accuracy of detection mechanisms. Additionally, integration with existing security systems and comprehensive testing in real-world scenarios would enhance the applicability and effectiveness of the framework in practical settings.

Overall, the proposed detection framework for USB rubber duckies provides a promising step towards strengthening USB security and mitigating the risks posed by malicious HID devices, ultimately enhancing the protection of systems and data against potential BadUSB attacks.

Contributions:

Pranav Raj: Has contributed to the research article on the "Detection framework for USB rubber duckies" by focusing on the design and development of the detection algorithm. He has conducted an in-depth literature review to understand existing detection techniques and proposed an innovative approach based on analyzing the behavior patterns of HID devices. He was responsible for implementing the algorithm, conducting experiments, and analyzing the results. He has played a crucial role in fine-tuning the detection parameters to achieve high accuracy while minimizing false positives and false negatives.

Riya Vaid: Has contributed to the research article by focusing on the data collection and analysis aspects of the detection framework. She has designed and implemented the data collection module, which involved monitoring USB traffic and capturing relevant information from connected HID devices. She also devised the data analysis methodology, performing statistical analysis and machine learning techniques to extract meaningful patterns and features that distinguish USB rubber duckies from legitimate HID devices. Her contribution significantly enhanced the framework's ability to accurately detect potential threats.

Saniel Bhattarai : His contribution to the research article on the "Detection framework for USB rubber duckies" involved a concentrated effort on the design and development of the detection algorithm. They extensively reviewed existing detection techniques, gaining a deep understanding of the subject matter. Their proposed approach for analyzing HID device behavior patterns was innovative and formed a cornerstone of the research. They were actively involved in implementing the algorithm, conducting experiments, and meticulously analyzing the obtained results. Their role was pivotal in refining the detection parameters to achieve optimal accuracy, with a focus on minimizing both false positives and false negatives.

Together, the contributions of Saniel Bhattarai, Pranav Raj, and Riya Vaid formed a comprehensive research article that presented a robust and effective detection framework for USB rubber duckies. Their combined efforts in research, testing, and practical implementation resulted in comprehensive

research paper that addresses the critical need for enhanced USB security in the face of evolving threats.