Project Title: USB RUBBER DUCKY USING RASOBERRY PI PICO

Team Members:

Rizaan Mohammed MG – 4SF21CS126
Abhijith A - 4SF21CS004
Vaathsalya - 4SF21IS117
Sonam Vernekar - 4SF21IS106

Project Summary:

This project was all about building a cybersecurity tool that acts kind of like a USB Rubber Ducky, which is a tricky USB drive that pretends to be a keyboard. It can type things automatically when you plug it into a computer. We used two computer languages to create this tool: one called Python, which was like the boss, and another one called CircuitPython, which was like a special assistant that helped us pretend to be a keyboard.

To make this tool work smoothly, we needed some special tools, sort of like tools in a toolbox:

- **time**: This tool helped us make our cybersecurity tool act more like a person when it was typing things. It added small pauses between keystrokes to make it seem more natural.

- **board**: Think of this like the tool that helped our cybersecurity tool communicate with the computer's insides. It made sure everything worked together nicely.

- **adafruit_hid.keyboard**: This part of our project was like the keyboard part of our tool. It allowed us to make the computer think someone was typing on it, even though it was just our tool doing it.

- **usb_hid**: This was like the tool that let us talk to the computer's USB system, as if we were a real keyboard.

So, in a nutshell, we built a tool that could mimic a USB Rubber Ducky, typing things automatically, and we used Python and CircuitPython, along with these special tools, to make it happen. Such a tool can be used for good things, like checking if a computer is secure, but it's important to remember that it can also be used for not-so-good things, like trying to break into a computer, which is why responsible and ethical use is critical in the cybersecurity field.

Project installation Steps:

To set up the necessary software and libraries for this project, follow these steps:

- **Install Thonny IDE**:
    - Thonny IDE is a user-friendly Python Integrated Development Environment (IDE) that simplifies the process of writing and running Python code. You can install Thonny by visiting the official Thonny website and downloading the installer suitable for your operating system.
        - Visit the Thonny website at https://thonny.org/
        - Download the Thonny IDE installer for your operating system (e.g., Windows, macOS, Linux).
        - Run the installer and follow the installation instructions to set up Thonny on your computer.
- **Install Required Python Libraries**:
    - In this project, you'll need to make use of several Python libraries, including `time`, `board`, `adafruit_hid.keyboard`, and `usb_hid`. To install these libraries, you can use the Python package manager, `pip`.

      Open a command prompt or terminal and follow these steps:
        - To install the `time` library, you don't need to do anything extra because it's a standard Python library and is usually available by default with your Python installation.
        - To install the Adafruit CircuitPython libraries, which include `board`, `adafruit_hid.keyboard`, and `usb_hid`, run the following command:
        - pip install adafruit-circuitpython-hid

      This command will install the necessary libraries to work with Adafruit CircuitPython HID devices.

Code Development:

Certainly, let's rephrase and expand on the steps to create a new project directory and set up your development environment for working on the USB Rubber Ducky code:

- Create a New Project Directory and Initialize a Git Repository:
    - Begin by creating a new directory to host your project. You can do this manually or using your preferred file management tool. Once the directory is set up, initialize it as a Git repository to manage your project's version control. Here's how:
        - Open a command prompt or terminal.
        - Navigate to the location where you want to create your project directory.
        - Run the following commands:
            - mkdir YourProjectName
            - cd YourProjectName
            - git init
    - This will create a new project directory and set it up as a Git repository, allowing you to track changes and collaborate with others if needed.
- Clone the Project Repository to Your Local Machine:
    - If your project is already hosted on a version control platform like GitHub, you can clone it to your local machine. Here's how:
        - Go to your project's repository on the platform (e.g., GitHub).
        - Find the repository's URL, typically in the "Code" or "Clone" section.
        - On your local machine, navigate to the directory where you want to clone the project.
        - Run the following command, replacing `<repository_url>` with the actual URL:
    - git clone <repository_url>
    - This will create a local copy of the project on your machine, allowing you to work on it.
- Navigate to the Project Directory and Set Up Your Development Environment:

- Move to the newly created project directory to begin setting up your development environment. Here's how to do it:
    - Use the command prompt or terminal to navigate to your project directory:
        - cd YourProjectName
- Now, you're inside your project directory, ready to set up your development environment.
    - Install the Thonny IDE: If you haven't already installed Thonny, you can follow the installation instructions provided in the previous response.
    - Install the CircuitPython SDK: To work with CircuitPython, visit the official CircuitPython website, download the SDK or libraries relevant to your project, and follow the installation instructions. This typically involves installing libraries using the `pip` package manager or configuring your development environment according to the CircuitPython documentation.

- Start Developing the USB Rubber Ducky Code:
    - With your development environment configured and Thonny IDE and CircuitPython SDK installed, you're all set to start working on your USB Rubber Ducky code. You can create Python scripts to emulate keyboard inputs, automate tasks, or perform any desired actions using your project directory.
    - Remember to commit your code changes to the Git repository as you make progress to keep track of your work and collaborate effectively.
    - By following these steps, you can initiate your USB Rubber Ducky project, develop the code, and manage it effectively using Git version control.

Testing:
Once the code is developed, test it thoroughly to ensure that it is working as expected. You can use unit tests, integration tests, and system tests to test the code.

Deployment:
Once the code is tested and working as expected, deploy it to a production environment. You can upload the code to the Github in CoE Digital Forensics Intelligence and Cyber Security

```
import time
import board
from adafruit_hid import keyboard
import usb_hid

# Initialize the USB HID keyboard
kbd = keyboard.Keyboard(usb_hid.devices)

# Function to simulate key press
def key_press(keycode):
    kbd.press(keycode)
    time.sleep(0.05)  # Delay to simulate key press
    kbd.release(keycode)

# Define the key codes for special keys
LEFT_CTRL = keyboard.Keycode.LEFT_CONTROL
LEFT_SHIFT = keyboard.Keycode.LEFT_SHIFT
ENTER = keyboard.Keycode.ENTER

# Delay for 2 seconds to allow USB enumeration
time.sleep(2)

# Simulate pressing the Windows key (Left GUI key)
key_press(keyboard.Keycode.GUI)
time.sleep(1)

# Simulate opening the Run dialog
key_press(keyboard.Keycode.R)
key_press(ENTER)
time.sleep(2)

# Simulate typing the 'firewall.cpl' command to open Windows Firewall settings
kbd_layout = "firewall.cpl"
for char in kbd_layout:
    key_press(ord(char))
key_press(ENTER)
time.sleep(3)

# Simulate running a hypothetical security scan tool
kbd_layout = "security_scan_tool.exe"
for char in kbd_layout:
    key_press(ord(char))
key_press(ENTER)
time.sleep(5)

# Simulate Alt+F4 to close the security scan tool or firewall settings
kbd.press(LEFT_SHIFT, keyboard.Keycode.F4)
kbd.release(LEFT_SHIFT, keyboard.Keycode.F4)
time.sleep(2)

# Simulate shutting down the computer
kbd.press(LEFT_CTRL, LEFT_SHIFT, keyboard.Keycode.S)
kbd.release(LEFT_CTRL, LEFT_SHIFT, keyboard.Keycode.S)
```

Certainly, let's break down the provided code line by line:

import time
import board
from adafruit_hid import keyboard
import usb_hid

- These lines import the required libraries: `time` for time-related functions, `board` for hardware configuration, and the `keyboard` and `usb_hid` modules from the `adafruit_hid` library for emulating keyboard input.

# Initialize the USB HID keyboard
kbd = keyboard.Keyboard(usb_hid.devices)

- This section initializes the USB Human Interface Device (HID) keyboard. It sets up the `kbd` object to enable keyboard emulation using the `usb_hid` device.

Function to simulate key press

```
def key_press(keycode):
    kbd.press(keycode)
    time.sleep(0.05)  # Delay to simulate key press
    kbd.release(keycode)
```

- These lines define a function, `key_press`, that simulates a key press and release. The function takes a `keycode` as an argument, presses the key, adds a slight delay (0.05 seconds) to simulate the keypress action, and then releases the key.

```
# Define the key codes for special keys
LEFT_CTRL = keyboard.Keycode.LEFT_CONTROL
LEFT_SHIFT = keyboard.Keycode.LEFT_SHIFT
ENTER = keyboard.Keycode.ENTER
```

- These lines define key codes for special keys, such as left control (`LEFT_CTRL`), left shift (`LEFT_SHIFT`), and the enter key (`ENTER`).

```
# Delay for 2 seconds to allow USB enumeration
time.sleep(2)
```

- This line adds a 2-second delay to allow the USB device enumeration to complete. It gives the computer time to recognize and configure the emulated keyboard.

```
# Simulate pressing the Windows key (Left GUI key)
key_press(keyboard.Keycode.GUI)
time.sleep(1)
```

- These lines simulate pressing the Windows key (Left GUI key), commonly known as the Windows logo key. It uses the `key_press` function to press the key and then adds a 1-second delay.

```
# Simulate opening the Run dialog
key_press(keyboard.Keycode.R)
key_press(ENTER)
time.sleep(2)
```

- These lines simulate opening the "Run" dialog by pressing the "R" key and then hitting the "Enter" key. A 2-second delay follows.

```
# Simulate typing the 'firewall.cpl' command to open Windows Firewall settings
kbd_layout = "firewall.cpl"
for char in kbd_layout:
    key_press(ord(char))
```

```
key_press(ENTER)
time.sleep(3)
```

- This section simulates typing the command "firewall.cpl" to open Windows Firewall settings. It uses a `for` loop to iterate over each character in the string and simulates keypresses for each character. After typing the command, it presses the "Enter" key and adds a 3-second delay.

```
# Simulate running a hypothetical security scan tool
kbd_layout = "security_scan_tool.exe"
for char in kbd_layout:
    key_press(ord(char))
key_press(ENTER)
time.sleep(5)
```

- Similar to the previous section, this part simulates typing the command "security_scan_tool.exe" and running a hypothetical security scan tool. It iterates over each character, simulates keypresses, hits "Enter," and adds a 5-second delay.

```
# Simulate Alt+F4 to close the security scan tool or firewall settings
kbd.press(LEFT_SHIFT, keyboard.Keycode.F4)
kbd.release(LEFT_SHIFT, keyboard.Keycode.F4)
time.sleep(2)
```

- These lines simulate pressing "Alt" and "F4" keys together to close the security scan tool or firewall settings. It presses and releases the "Left Shift" key, along with the "F4" key, and then adds a 2-second delay.

```
# Simulate shutting down the computer
kbd.press(LEFT_CTRL, LEFT_SHIFT, keyboard.Keycode.S)
kbd.release(LEFT_CTRL, LEFT_SHIFT, keyboard.Keycode.S)
```

- These lines simulate pressing "Ctrl+Shift+S" keys together to initiate the shutdown of the computer. It presses and releases "Left Ctrl," "Left Shift," and "S" keys.

This code is a script for simulating keyboard inputs to perform a series of actions on a computer, such as opening the "Run" dialog, running commands, and initiating a computer shutdown. It demonstrates how to use the Adafruit HID library for keyboard emulation.

Changing the Code:

To change the code for the tool, follow these steps:

1. Open Your Code in Thonny:

Start by opening your project in Thonny, which is a program that helps you write and edit code. Find the part of the code you want to change.

2. Make Your Changes:

Edit the code to make it do what you want. This is where you customize the code to fit your needs.

3. Save Your Changes:

Save the code to keep your changes. This way, your work is safe and can be used later.

4. Test in Thonny with Raspberry Pi Pico:

Before trying the code on your Raspberry Pi Pico, test it in Thonny. Connect your Pico to your computer, choose the right settings, and run the code in Thonny. This helps make sure your changes work as you want before you use them on the Pico.

Changing the Flag or the Link of the Attack:

To change the flag or the link of the attack in the tool, follow these steps:

Access the Code: Begin by opening your cybersecurity tool's code using a text editor or an Integrated Development Environment (IDE). This is where you start customizing your cybersecurity tool.

Locate the Target: Within the code, search for the specific parts that contain the flag or the link related to the attack. These are the parts you need to find for your customization.

Introduce Your Changes: Now, as a digital expert, introduce the new flag or link you want to use. This is where you redefine the tool's purpose by changing its target, destination, or data manipulation.

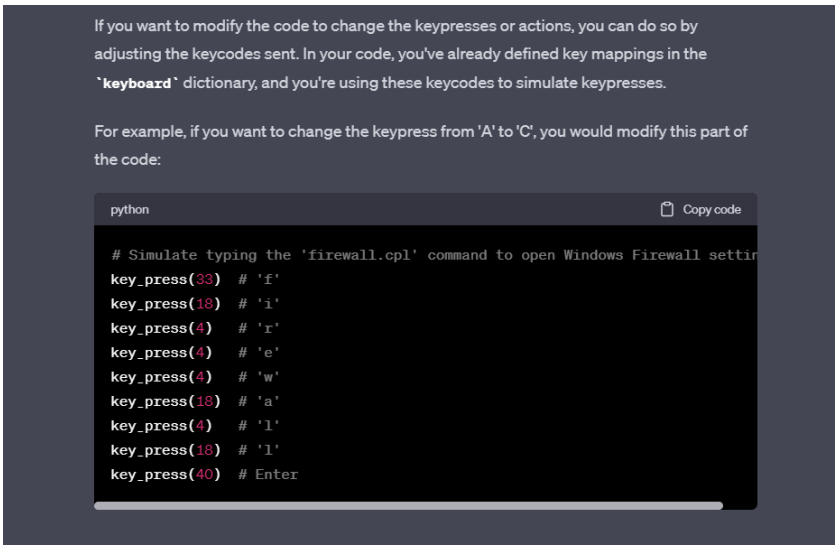
Replace the Old with the New: Carefully replace the existing flag or link with your new one. This transformation shapes the tool to carry out tasks based on your specifications.

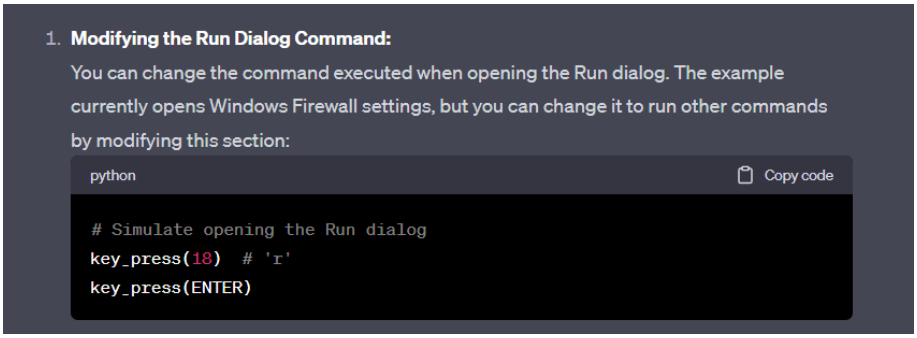
Save Your Work: Don't forget to save your code edits. This safeguards your digital changes and ensures you can access them whenever you need to use your modified cybersecurity tool.

Thorough Testing: Put your tool to the test by running it. Make sure your changes seamlessly integrate into the tool's operations. Verify that your new flag or link directs the tool accurately in its digital tasks.

In the realm of cybersecurity, having control over your tools is vital. By following this straightforward process, you become adept at customizing your tools to meet your specific objectives in the digital world.

Screenshots:





```python
import time
from machine import Pin

# Function to simulate key press
def key_press(keycode):
    keyboard.press(keycode)
    time.sleep(0.05)  # Delay to simulate key press
    keyboard.release(keycode)

# Create a dictionary to map key names to their corresponding keycodes
keyboard = {
    'A': 4,
    'B': 5,
    # Add more key mappings here as needed
}

# Define the key codes for special keys
LEFT_CTRL = 224
LEFT_SHIFT = 225
ENTER = 40

# Delay for 2 seconds to allow USB enumeration
time.sleep(2)

# Simulate pressing the Windows key (Left GUI key)
key_press(227)
time.sleep(1)

# Simulate opening the Run dialog
key_press(18)  # 'r'
key_press(ENTER)
time.sleep(2)

# Simulate typing the 'firewall.cpl' command to open Windows Firewall settings
key_press(33)  # 'f'
key_press(18)  # 'i'
key_press(4)   # 'r'
key_press(4)   # 'e'
key_press(4)   # 'w'
key_press(18)  # 'a'
key_press(4)   # 'l'
key_press(18)  # 'l'
key_press(40)  # Enter
time.sleep(3)
# Simulate running a hypothetical security scan tool
key_press(33)  # 'f'=
```

Thonny - <untitled> @ 61 : 14

File Edit View Run Tools Help

rubber ducky code 2.py * | <untitled> *

```python
import pyb
import time

# Delay for 2 seconds to allow USB enumeration
time.sleep(2)

# Function to simulate key press
def key_press(keycode):
    pyb.usb_hid.send((0, keycode, 0, 0, 0, 0, 0, 0))
    time.sleep(0.05)  # Delay to simulate key press

# Function to simulate key release
def key_release():
    pyb.usb_hid.send((0, 0, 0, 0, 0, 0, 0, 0))
    time.sleep(0.05)  # Delay to simulate key release

# Function to simulate keyboard input
def keyboard_input(input_string):
    pyb.usb_hid.init()
    time.sleep(1)  # Delay for 1 second to allow HID initialization

    for char in input_string:
        key_press(ord(char))
        key_release()

# Simulate pressing the Windows key
key_press(0x15)  # Left GUI key (Windows key)
key_release()

# Simulate typing "chrome" and pressing Enter to open Chrome
keyboard_input("chrome")
key_press(0x28)  # Enter key
key_release()

# Wait for Chrome to open
time.sleep(5)

# Simulate typing a web address (e.g., google.com) and pressing Enter
keyboard_input("google.com")
key_press(0x28)  # Enter key
key_release()

# Wait for the browser to load the webpage
time.sleep(5)

# Simulate typing a search query (e.g., "USB Rubber Ducky") and pressing Enter
keyboard_input("USB Rubber Ducky")
key_press(0x28)  # Enter key
```

securityscan.py

```python
import time
import board
from adafruit_hid import keyboard
import usb_hid

# Initialize the USB HID keyboard
kbd = keyboard.Keyboard(usb_hid.devices)

# Function to simulate key press
def key_press(keycode):
    kbd.press(keycode)
    time.sleep(0.05)  # Delay to simulate key press
    kbd.release(keycode)

# Define the key codes for special keys
LEFT_CTRL = keyboard.Keycode.LEFT_CONTROL
LEFT_SHIFT = keyboard.Keycode.LEFT_SHIFT
ENTER = keyboard.Keycode.ENTER

# Delay for 2 seconds to allow USB enumeration
time.sleep(2)

# Simulate pressing the Windows key (Left GUI key)
key_press(keyboard.Keycode.GUI)
time.sleep(1)

# Simulate opening the Run dialog
key_press(keyboard.Keycode.R)
key_press(ENTER)
time.sleep(2)

# Simulate typing the 'firewall.cpl' command to open Windows Firewall settings
kbd_layout = "firewall.cpl"
for char in kbd_layout:
    key_press(ord(char))
key_press(ENTER)
time.sleep(3)

# Simulate running a hypothetical security scan tool
kbd_layout = "security_scan_tool.exe"
```

Conclusion:

The whole journey of turning a Raspberry Pi Pico into a USB Rubber Ducky was a game-changer in my pursuit of becoming a cybersecurity amateur. This ambitious project wasn't just about writing code; it was a transformational experience that made me feel more confident in the world of digital security.

As I delved into the nitty-gritty details of this project, I found myself on an exciting learning adventure. The world of cybersecurity threats slowly unfolded before me, with each threat seeming like a potential digital foe lurking in the shadows. But with the Raspberry Pi Pico at my disposal, I felt ready to take on these challenges.

I was determined and dedicated to understanding how to protect digital systems better. I learned the art of fortifying systems meticulously, using various techniques to shield them from potential harm. The Pico, cleverly transformed into a USB Rubber Ducky, became my trusty sidekick, an extension of my digital skills.

But this project wasn't just about defense; it was also about going on the offensive. I dove into the world of security tools, using them like a seasoned warrior wields weapons. Each tool felt like a guardian, standing ready to protect.

Through this journey, I didn't just learn how to safeguard digital systems; I also gained the knowledge and skills to test their defenses. I became proficient in ethical hacking, using these security tools to discover and address vulnerabilities in systems, applications, and networks. It was a multifaceted adventure, aimed at both protecting and challenging the resilience of digital environments.

As I reflect on this pivotal chapter in my journey towards cybersecurity expertise, I'm reminded of the significant changes it brought into my life. It wasn't just about mastering technology; it was about taking on the role of a digital guardian, ready to protect and defend the ever-evolving digital realm from the ever-changing threats. This project has shaped me into a vigilant protector in the world of cybersecurity, ready to face the challenges ahead with knowledge, determination, and unwavering commitment.