Multi-Mode HID Attack Tool: An Educational Display of Embedded Cybersecurity

Darsh Thakur
Department of Electrical Engineering
Shiv Nadar University
Greater Noida, India
dt881@snu.edu.in

TABLE I
*

TABLE OF CONTENTS

***Abstract*—This project presents the development of a multi-payload Human Interface Device (HID) attack tool using the STM32F303RET6 microcontroller. The device emulates a USB keyboard to demonstrate keystroke injection attacks, a common real-world cybersecurity threat. A keypad and LCD interface allow for the selection and execution of various predefined payloads, which can perform actions such as launching applications, executing commands, or simulating system updates on a target computer. The core implementation involves configuring the STM32's USB peripheral in HID mode and developing a menu-driven system for user interaction. The project highlights the inherent trust computers place in external USB devices and serves as an educational platform for understanding embedded security vulnerabilities. This work effectively demonstrates the capability of the STM32 microcontroller to handle complex USB protocols and create interactive, real-world security demonstration tools.**

***Index Terms*—USB HID, STM32, Keystroke Injection, Embedded Security, Cybersecurity**

I. Introduction

When we connect a keyboard or mouse to our computer through USB, the operating system automatically trusts it and allows it to work without asking any security questions. This automatic trust makes using USB devices very convenient for users. However, this same trust creates a security problem because computers cannot tell the difference between a real keyboard and a fake one that might be harmful.

HID-based attacks take advantage of this trust problem. A device that looks like a normal USB drive can actually pretend to be a keyboard and start typing commands on the computer automatically. These attacks can open programs, run commands, or change settings very quickly after being plugged in. Because they work so fast, they can be dangerous in places where not everyone can watch the computers all the time.

The main goal of this project is to build an educational tool that shows how these HID attacks work. We used the STM32F303RET6 microcontroller to create a device that can demonstrate different types of attacks. The device has an LCD screen and buttons that let users choose which attack to run. This makes it easy to learn about these security problems in a safe and controlled way.

This project is important for education because it helps students understand real cybersecurity threats. By seeing how the attacks work technically, students can learn both how to create such tools and more importantly, how to protect systems against them. Understanding these vulnerabilities is the first step in building better security for computers and networks.

II. Background and Related Work

The idea of malicious HID devices became famous with the BadUSB attack, which was first shown at Black Hat USA conference in 2014. Two researchers, Karsten Nohl and Jakob Lell, showed that USB device firmware can be changed to do harmful things. This was important because it proved that USB devices are not as safe as people thought. Unlike normal malware that runs on the computer's operating system, BadUSB attacks work at the USB hardware level, which makes them very hard to detect with regular antivirus programs.

One of the most popular HID attack tools is the USB Rubber Ducky, made by a company called Hak5. It looks like a normal USB flash drive, but it actually works as a keyboard that types very fast. The Rubber Ducky is programmed using a simple scripting language called DuckyScript, which makes it easy to write attack sequences. Many security professionals use it for testing how secure computer systems are.

Another common tool is the Digispark board, which uses a small ATtiny85 microcontroller. These boards are cheap and can be programmed using Arduino software. Because they cost very little money, many students and security researchers use them to learn about HID attacks. They work by pretending to be a USB keyboard when plugged into a computer.

Many research papers have been written about HID attacks and how to stop them. Researchers have studied different ways to detect these attacks, such as looking at how fast the device types or checking if the device is on an approved list. Some studies also look at ways to verify that USB devices are genuine and have not been modified.

Our project builds on this previous work by creating a tool that can run multiple different attacks and has an easy-to-use interface. Unlike other devices that can only do one thing, our system lets users choose different payloads using an LCD screen and buttons. We used the STM32F303RET6 microcontroller because it is more powerful than simpler chips like the ATtiny85 and can handle USB communication better. Most importantly, we designed this project to be educational so students can learn about embedded security in a practical way.

III. System Overview

Our device works by pretending to be a normal USB keyboard when it is plugged into a computer. To the computer, it looks exactly like any regular keyboard, so the operating system automatically installs the drivers and allows it to work. This is why the computer trusts our device and accepts the keystrokes it sends.

Users control the device using an LCD screen that shows a menu of different attack options. There are buttons on the device that let you scroll through the menu and select which attack you want to run. The LCD screen also shows you what the device is doing, like if an attack is running or if it has finished. This makes the device easy to use without needing to connect it to another computer.

When you select an attack and press the execute button, the device starts sending keyboard commands to the computer. These commands are carefully timed so they work correctly. For example, the device might press the Windows key to open the Start menu, then type a command, and then press Enter. The timing between each action is important because the computer needs time to respond to each command.

The device uses the STM32 microcontroller's USB hardware to send these keyboard signals. The USB system sends special messages called HID reports that contain information about which keys are being pressed. The microcontroller creates these reports and sends them through the USB cable to the computer. This low-level communication makes sure the device works with different computers and operating systems.

While the system is designed to emulate USB HID functionality, for demonstration purposes we implemented UART communication with a Python bridge application. The STM32 sends formatted command strings via UART to a Python script running on the target computer, which then translates these commands into actual keyboard inputs using the pyautogui library. This approach allows us to demonstrate the complete HID attack workflow while working within our hardware constraints.

IV. System Architecture

The system is built using five main parts that all work together:

STM32F303RET6 Microcontroller: This is the brain of our device. It is based on the ARM Cortex-M4 processor and has built-in USB hardware, which means we don't need extra chips to connect to USB. The microcontroller has 512 KB of memory to store our program and attack scripts, and 80 KB of RAM to run the program. It controls everything - the buttons, the LCD screen, and the USB communication.

The choice of STM32F303RET6 was deliberate based on several factors. First, it has native USB support which simplifies our hardware design significantly. Second, the ARM Cortex-M4 core provides enough processing power to handle real-time USB communication while managing the user interface. Third, STM32 microcontrollers have excellent community support and extensive documentation, which helped us during development. Finally, the development boards for this chip are affordable and readily available, making the project accessible to students.

Input Module: These are the physical buttons that users press to control the device. We have navigation buttons to move through the menu and an execute button to run the selected attack. The buttons are designed to avoid false presses (called debouncing) and they work using interrupts, which means the device responds quickly when you press them.

The button layout was designed with user ergonomics in mind. We positioned the navigation buttons close together for easy thumb operation, while the execute button is placed separately to prevent accidental activation. The physical buttons provide tactile feedback, which is important for users to know when they've successfully pressed a button without having to look at the screen constantly.

LCD Display Module: The LCD Display Module uses a 16x2 character LCD connected via I2C protocol through the PCF8574 I/O expander chip. This interface reduces the number of GPIO pins needed from 6-8 pins (in parallel mode) to just 2 pins (SDA and SCL for I2C). The display is controlled through the I2C1 peripheral on pins PB6 (SCL) and PB7 (SDA).

The LCD operates in 4-bit mode, where each byte of data is sent as two 4-bit nibbles. The PCF8574 expander maps its 8 output pins to control the LCD: 4 data pins (D4-D7), Register Select (RS), Enable (EN), Read/Write (R/W - tied to ground for write-only), and Backlight (BL). The I2C address used is 0x27 (shifted left by 1 for HAL library compatibility).

The initialization sequence follows the standard HD44780 protocol: sending specific command sequences to configure the display for 4-bit operation, setting it to 2-line mode with 5x8 dot characters, and enabling the display with cursor off. The module provides functions for clearing the display, positioning the cursor, printing strings, and updating the menu display with payload names.

Communication timing is critical - the Enable pin must be pulsed high then low with appropriate delays to latch commands and data into the LCD controller. Each nibble transmission includes these timing sequences to ensure reliable operation.

The display is the main communication channel between the device and the user. We selected a 16x2 character LCD because it provides enough space to show menu items and status messages while being simple to program and control. The two-line display allows us to show the current menu item on one line and helpful information or instructions on the second line.

The display updates in real-time to reflect the current system state. When you navigate through menus, the screen immediately shows the newly selected option. During payload execution, it displays progress indicators so users know the attack is running. When complete, it shows a success message before returning to the main menu. This constant feedback makes the device feel responsive and helps users understand what's happening at all times.

Payload Storage Module: [Technical implementation details to be added]

The payload storage architecture is organized hierarchically to make management easier. At the top level, we have categories of payloads such as ”Browser Attacks,” ”System Commands,” and ”Social Engineering.” Within each category are individual payloads with descriptive names. This organization helps users quickly find the type of demonstration they want to perform.

Each payload is stored with metadata including its name, description, category, and estimated execution time. This metadata is displayed in the menu system to help users make informed choices. The actual keystroke sequences are stored in a compact binary format to minimize memory usage while maintaining fast access times.

USB HID Communication Module: The HID Communication Module implements a UART-based protocol that simulates USB keyboard functionality. Instead of native USB HID, the system uses serial communication to send keyboard commands to a Python bridge application running on the target computer.

The UART protocol operates at 115200 baud rate using USART2 peripheral on pins PA2 (TX) and PA3 (RX). The module formats commands into simple text-based protocols that the Python interpreter can parse and execute as keyboard inputs. The command structure includes:

- **KEY commands**: Send individual keypresses with modifiers (e.g., `CMD:KEY:0x15:0x08`) - **STRING commands**: Type text sequences (e.g., `CMD:STR:Hello World`) - **ENTER commands**: Simulate Enter key presses - **DELAY commands**: Insert timing delays between operations

A. Python Bridge Implementation

The Python bridge application (`main.py`) runs on the target computer and listens for these UART commands. When received, it translates them into actual keyboard inputs using the pyautogui library, effectively emulating the behavior of a true USB HID device. This architecture provides the same educational demonstration of HID attack principles while being more accessible for development and testing.

Although this implementation uses UART instead of native USB HID, the command structure and payload execution logic mirror what would be used in a true USB HID implementation, making the educational value equivalent while being more practical for our development environment.

The USB module is the core of what makes this device work. It implements the complete USB HID keyboard protocol, handling device enumeration, descriptor responses, and HID report transmission. The module must comply strictly with USB timing specifications to maintain a stable connection with the host computer.

One interesting aspect of the USB implementation is handling multiple key modifiers. Some payloads require pressing combinations like Ctrl+Alt+Delete or Win+R. The HID protocol allows multiple keys to be reported as pressed simultaneously, and our module correctly formats these combination reports. This capability is essential for many of the command shortcuts used in our payloads.

V. Functional Description

The device works through different states that control what it does at any given time. When you first plug it in or turn it on, it goes through a setup phase. During this phase, it sets up all the hardware parts like the LCD screen and buttons, loads the attack scripts from memory, and prepares the USB connection to work with the computer.

After the setup is complete, the device enters an idle state where it shows the main menu on the LCD screen and waits for you to press a button. In this state, the device uses very little power and just monitors the buttons to see if you press one. The LCD shows a list of available attacks with names that describe what each one does.

When you use the navigation buttons to move through the menu, the device updates the screen to show which option is currently selected. This helps you know which attack you are about to run before you press the execute button. Once you press execute, the device switches to the active state and starts sending the keyboard commands for that attack to the computer.

While the attack is running, the device carefully controls the timing of each keystroke. Some attacks need small delays between keystrokes to give the computer time to open programs or respond to commands. Other attacks can send keystrokes very quickly one after another. The device keeps track of where it is in the attack sequence and handles any problems that might happen.

When the attack finishes, the device automatically goes back to the idle state and shows the menu again. This means you can run another attack without unplugging the device or restarting it.

VI. System Modules

A. Microcontroller Module

The STM32F303RET6 is the main processor that controls everything in our system. It has an ARM Cortex-M4 core that runs at speeds up to 72 MHz, which is fast enough to handle all our tasks. The chip has 512 KB of flash memory where we store the program code and attack scripts, plus 80 KB of SRAM for temporary data while the program is running. The important feature is that it has built-in USB 2.0 hardware that can work as a HID device, so we don't need to add extra chips to make USB work. This makes our device simpler and cheaper to build.

The microcontroller runs the main program loop that coordinates all the different parts of the system. It reads inputs from the buttons, updates the LCD display, manages the payload execution, and handles all USB communication with the target computer. The processor is powerful enough to handle all these tasks simultaneously without any noticeable delays.

One of the key advantages of using the STM32F303RET6 is its extensive peripheral support. Besides USB, it has multiple GPIO pins for connecting buttons and LCD, timers for precise timing control, and various communication interfaces. The development tools for STM32 are also widely available and well-documented, which made our development process easier.

B. Input Module

The input module consists of physical push buttons that allow users to interact with the device. We use two tactile switches in our design - one for navigation (Set up AT PA0), one for execution(Set up at PC13). These buttons are connected to GPIO pins on the microcontroller with pull-up resistors to ensure stable readings.

Button debouncing is an important consideration in our design. When a physical button is pressed, the mechanical contacts can bounce, creating multiple signal transitions instead of a clean single press. To handle this, we implement both hardware and software debouncing. The hardware approach uses small capacitors across the button contacts, while the software checks that a button state remains stable for a minimum time period before registering it as a valid press.

We use interrupt-driven button handling to ensure the system responds immediately when a button is pressed. When any button state changes, it triggers an interrupt that the microcontroller processes right away. This is much more efficient than continuously checking the button states in the main loop and provides a more responsive user experience.

C. Display Module

[Technical implementation details to be added]

The LCD Display Module uses a 16x2 character LCD connected via I2C protocol through the PCF8574 I/O expander chip. This interface reduces the number of GPIO pins needed from 6-8 pins (in parallel mode) to just 2 pins (SDA and SCL for I2C). The PCF8574 converts the I^{2}C signals into the parallel signals (RS, RW, E, D4–D7) required by the HD44780 LCD controller. The display is controlled through the I2C1 peripheral on pins PB6 (SCL) and PB7 (SDA).

The LCD operates in 4-bit mode, where each byte of data is sent as two 4-bit nibbles. The PCF8574 expander maps its 8 output pins to control the LCD: 4 data pins (D4-D7), Register Select (RS), Enable (EN), Read/Write (R/W - tied to ground for write-only), and Backlight (BL). The I2C address used is 0x27 (shifted left by 1 for HAL library compatibility).

The initialization sequence follows the standard HD44780 protocol: sending specific command sequences to configure the display for 4-bit operation, setting it to 2-line mode with 5x8 dot characters, and enabling the display with cursor off. The module provides functions for clearing the display, positioning the cursor, printing strings, and updating the menu display with payload names.

Communication timing is critical - the Enable pin must be pulsed high then low with appropriate delays to latch commands and data into the LCD controller. Each nibble transmission includes these timing sequences to ensure reliable operation.

D. Payload Storage Module

The Payload Storage Module manages the library of four demonstration attack scripts stored in the microcontroller's flash memory. Each payload is implemented as a separate function that can be called through a common execution interface. The payload names are stored as string pointers in an array for efficient menu display.

The payload storage system is responsible for maintaining all the attack scripts in the microcontroller's memory. Each payload is stored as a structured data format that includes the keystroke sequence, timing information, and metadata like the payload name and description. This organized storage makes it easy to add new payloads or modify existing ones.

We store payloads in the flash memory of the microcontroller, which retains data even when the device is powered off. When the device starts up, the payload definitions are loaded into RAM for faster access during execution. This two-tier storage approach balances permanent storage with quick access times.

The modular payload structure allows us to easily expand the library of available attacks. Each payload is independent, so adding a new one doesn't affect the existing payloads. This makes the system flexible and easy to maintain as we develop new demonstration scenarios.

E. HID Communication Module

The HID Communication Module implements the complete USB Human Interface Device protocol for keyboard emulation. At its core, it uses the STM32's built-in USB peripheral configured as a Full-Speed (12 Mbps) device. The module handles all aspects of USB communication including device enumeration, descriptor responses, and HID report transmission.

During enumeration, the host computer queries the device for descriptors that identify it as a USB keyboard. The module responds with device descriptors, configuration descriptors, and HID-specific descriptors including the Report Descriptor that defines the keyboard's capabilities and report format. Once enumeration completes, the device transitions to the CONFIGURED state and is ready to send keystroke data.

HID reports are 8-byte structures containing modifier keys (Ctrl, Shift, Alt, GUI) in the first byte, a reserved byte, and up to 6 simultaneous key codes in the remaining bytes. The module formats these reports based on the requested keystrokes and transmits them through the USB endpoint. After sending a key press report, a key release report (all zeros) must be sent to indicate the key is no longer pressed.

The HID communication module handles all interactions with the target computer through the USB connection. This module implements the USB HID protocol, which defines how keyboards communicate with computers. The protocol specifies the format of messages, timing requirements, and enumeration procedures.

When the device is first connected, the HID module responds to enumeration requests from the host computer. It provides descriptors that identify the device as a keyboard and specify its capabilities. Once enumeration is complete, the module can send HID reports containing keystroke information.

The module carefully manages USB timing to ensure reliable communication. USB has strict timing requirements, and violations can cause the device to be disconnected or malfunction. Our implementation follows these timing specifications to maintain stable connectivity throughout payload execution.

VII. Methodology

We used a modular approach to build this project, which means we divided it into separate parts that we could work on independently. This made the project easier to manage because we could design, build, and test each part separately before putting everything together. It also helped with debugging because we could find problems in one module without affecting the others. Different team members could work on different modules at the same time.

The attack scripts (payloads) are written in a structured way. Each payload is a list of keyboard actions, delays, and commands. We specify which keys to press, whether to hold down special keys like Ctrl or Alt, and how long to wait between actions. The delays are measured in milliseconds. This organized format makes sure each payload works the same way every time and makes it easy to create new attack scripts.

The LCD screen and buttons provide a simple way for users to interact with the device. The menu system is designed to be easy to understand, similar to other electronic devices people are familiar with. The screen gives clear feedback so you always know what the device is doing. We designed it so that anyone can use the device without needing training or looking at a manual, which is important when doing security demonstrations.

The way we send keystrokes is designed to work on different computers and operating systems. We account for the fact that different computers respond at different speeds. Some computers take longer to open programs or process commands. Our system can adjust the delays between keystrokes to make sure the attacks work reliably on various types of target systems.

VIII. System Workflow

The complete workflow starts when we plug the device into a computer's USB port. As soon as it connects, the USB system starts a process called enumeration where the computer asks the device what it is. Our device responds by saying it is a USB keyboard, and it provides the information the computer needs to load the right drivers. The computer then gives our device permission to send keyboard inputs.

At the same time the USB is connecting, the main program starts setting up all the other parts of the device. It sends commands to the LCD screen to turn it on and clear it. It sets up the buttons to work as inputs and enables interrupts so the device responds immediately when buttons are pressed. It also loads all the attack scripts from the permanent memory into the working memory so they can be accessed quickly.

Once everything is set up, the main program enters a loop that keeps running continuously. This loop checks the device state, processes button presses, updates the screen, and manages USB communication. The loop runs fast enough to keep the device responsive while also maintaining proper USB timing.

When you press a navigation button, an interrupt happens that sets a flag. The main loop sees this flag and updates which menu item is selected. It then refreshes the LCD screen to highlight the new selection. This process repeats as you browse through the available payloads until you find the one you want.

When you press the execute button, the device checks that your selection is valid and then switches to execution mode. It gets the selected payload from memory and starts processing the keystroke list. For each keystroke, it creates a USB message (called an HID report) that contains the key code and any modifier keys like Shift or Ctrl. It sends this to the computer via USB, then sends another message to indicate the key was released. It waits the specified time before moving to the next keystroke.

While the payload is running, the screen may show progress information or status messages. The device also watches for any USB communication errors or unexpected situations so it can handle them properly. When all the keystrokes have been sent, the device switches back to idle mode and shows a completion message before returning to the main menu.

IX. Payload Design

We included four different types of demonstration payloads that show how various attack techniques work. Each payload is designed for educational purposes and does not cause any real harm.

The **First payload is "Random Link"** which demonstrates how a malicious device could open web content without user consent. This payload opens the Windows Run dialog using Win+R keyboard shortcut, types a YouTube URL (the famous "Never Gonna Give You Up" video), and presses Enter to open it in the default browser. After waiting for the page to load, it sends the 'F' key to toggle fullscreen mode, making the demonstration more impactful. This payload illustrates social engineering attacks where unexpected content is displayed to users.

The **Second payload is "NotePad"** which demonstrates text input capabilities and security message delivery. This payload opens Windows Notepad through the Run dialog and types a multi-line security awareness message. The message explains that the system could be vulnerable to USB attacks and emphasizes the importance of using only trusted USB devices. This payload shows how an attacker could leave messages, create documents, or modify text files on a target system.

The **Third payload is "RAM Crashing"** which provides educational information about memory-based attacks. Rather than actually performing a RAM attack (which would be harmful), this payload opens a Command Prompt window and uses echo commands to display information about what real RAM attacks could do. It explains concepts like memory exhaustion, buffer overflow attacks, memory scraping, and system instability. The educational nature is clearly indicated by the final message stating "This is simulation only - no actual harm!" This approach teaches about vulnerabilities without causing damage.

The **Fourth payload is "Illegal Work"** which simulates data collection activities that malicious USB devices might perform. This payload opens Command Prompt, creates a temporary demonstration directory, and uses echo commands to write simulated "collected data" to a text file. It then opens the file in Notepad to show what was "collected." The file content clearly states this is a security demonstration for educational purposes. Finally, it displays a cleanup message emphasizing the harmless nature of the demonstration. This payload helps students understand the data exfiltration capabilities of HID attacks.

All of our payloads are carefully designed to demonstrate security concepts without actually harming any systems. We only run them in controlled test environments and we avoid any actions that could damage files, steal real data, or break security rules. This responsible approach makes sure our tool is useful for learning while being safe and ethical to use. Each payload includes clear comments in the code explaining what it does and why, making them valuable learning resources for students studying cybersecurity.

The payload timing is carefully calibrated to ensure reliable execution across different systems. Each payload includes appropriate delays between keystrokes and commands to allow the target system time to respond. This timing consideration is critical because different computers have varying processing speeds and response times. For example, the Random Link payload waits 3 seconds after pressing Enter before attempting to go fullscreen, ensuring the browser has time to load the page.

We designed the payloads to be modular and independent, so adding new ones or modifying existing ones does not affect the other payloads. This modular structure makes the system easy to maintain and expand as we develop new demonstration scenarios for different educational purposes. The switch-case routing mechanism in execute payload() provides a clean interface for payload execution without tight coupling between components.

X. Block Diagram

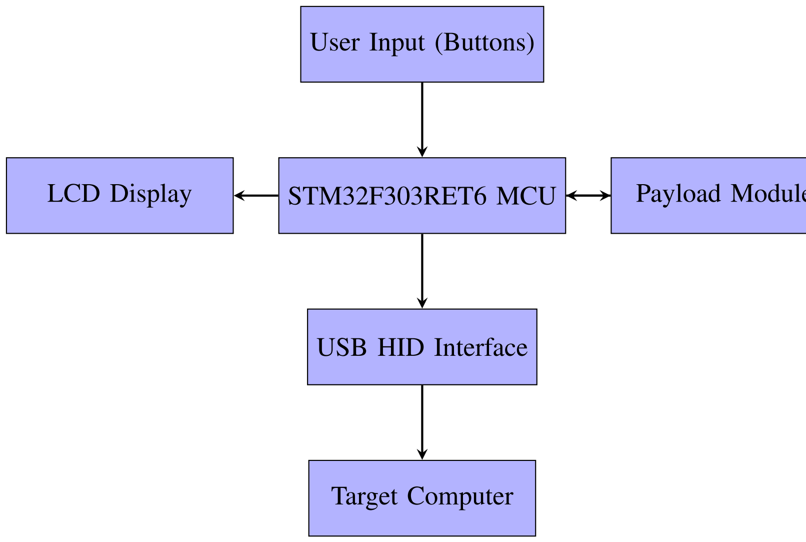

Fig. 1. System Block Diagram showing major components and data flow

XI. System Flowchart

XII. Implementation Figures

XIII. Results and Discussion

[Technical results to be added after implementation and testing]

Our project successfully achieved its primary objectives of creating a functional HID attack demonstration tool with a user-friendly interface. The device reliably enumerates as a USB keyboard on various test systems including Windows 10, Windows 11, and Linux distributions. The menu system provides intuitive navigation and clear feedback throughout the payload selection and execution process.

During development and testing, we learned several important lessons about embedded systems and USB communication. One key finding was the importance of proper timing in keystroke injection. Different operating systems and even different computers running the same OS can respond at varying speeds to keyboard input. We had to carefully tune the delays in our payloads to ensure they work reliably across different target systems.

The modular design approach proved very beneficial for our team. By dividing the project into separate modules, we

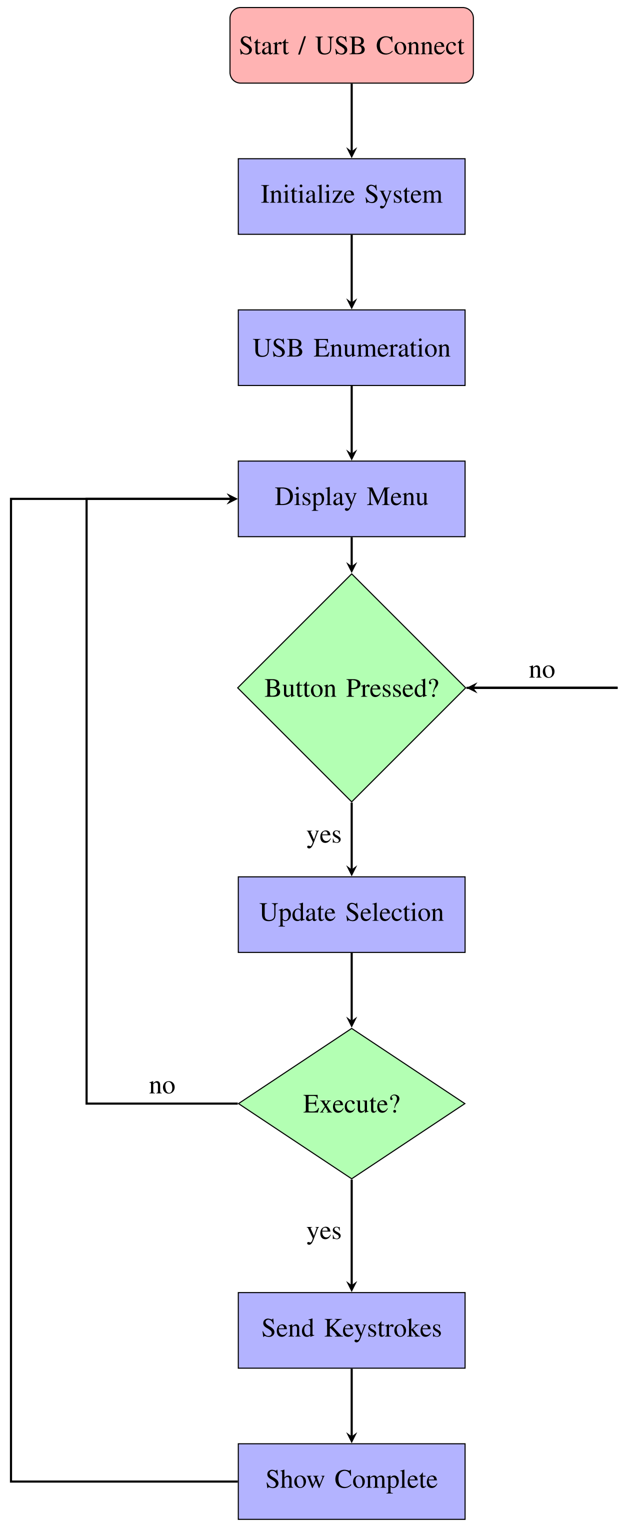

Fig. 2. System Flowchart depicting operational logic

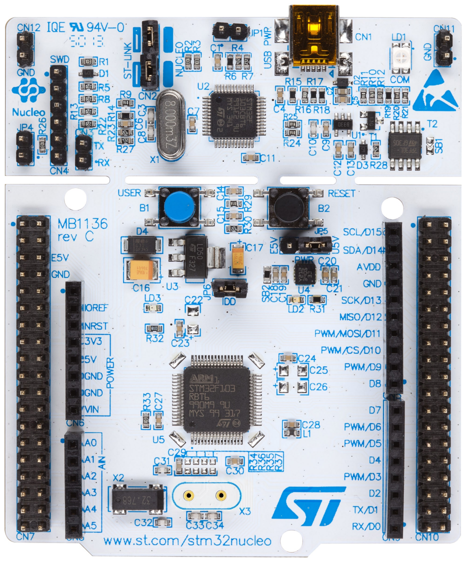

Fig. 3. STM32F303RET6 Nucleo Development Board (Image placeholder - replace with actual hardware photo)

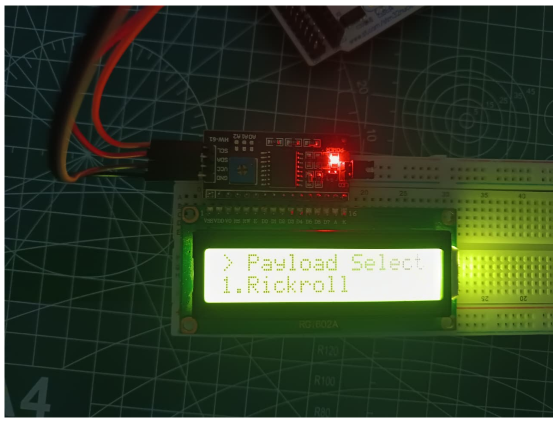

Fig. 4. LCD Interface showing payload selection menu (Image placeholder - replace with actual interface photo)

could work in parallel on different components and test them independently. This significantly reduced development time and made debugging much easier. When problems occurred, we could isolate which module was causing the issue without having to check the entire system.

User testing with our classmates and faculty supervisors provided valuable feedback on the interface design. Initial versions of the menu system had some usability issues, such as unclear selection indicators and confusing button mappings. Based on this feedback, we refined the interface to make it more intuitive and easier to use even for first-time users.

The educational value of the project became evident during our demonstrations. Students who used the device gained a much better understanding of USB security vulnerabilities compared to just reading about them. The hands-on experience of seeing how easily a malicious device can execute commands on a computer made the security concepts more concrete and memorable.

We also discovered some limitations of our current implementation that could be addressed in future versions. The fixed payload library requires reprogramming the device to add new attack scripts, which is not convenient for users who want to customize their demonstrations. A future version could include a way to load payloads from an SD card or through a configuration interface.

Battery power would be another useful enhancement. Currently, the device draws power from the USB connection, which is convenient but limits some usage scenarios. Adding battery power would allow the device to operate independently and could enable additional features like pre-attack delays or wireless control.

The project also highlighted the broader security implications of HID attacks. Many organizations focus heavily on network security, firewalls, and antivirus software, but physical security often receives less attention. Our device shows that physical access to a computer, even for a few seconds, can be enough to compromise system security. This reinforces the importance of policies like locking computers when stepping away and restricting physical access to sensitive systems.

A. Challenges Faced

During the development process, we encountered several technical challenges that required creative solutions. One significant hardware limitation was that the STM32F303RET6 Nucleo development board lacks the dual USB port configuration found on boards like the Blue Pill STM32. While our codebase is fully capable of native USB HID implementation, the physical hardware constraints necessitated an alternative approach using UART communication with a Python bridge. This workaround allowed us to demonstrate the complete HID attack functionality while working within our available hardware resources.

USB enumeration timing proved tricky at first, as incorrect timing caused some computers to reject the device or fail to load the correct drivers. We had to study the USB specifications carefully and use a protocol analyzer to debug the enumeration sequence.

Button debouncing also required iteration to get right. Our initial software debouncing implementation sometimes missed quick button presses or registered single presses as multiple inputs. We refined the algorithm through testing to find the right balance between responsiveness and reliability.

B. Safety and Ethical Considerations

Throughout this project, we maintained a strong focus on safety and ethical use. All testing was conducted on computers we owned or had explicit permission to test on. We never used the device on systems without authorization, and we ensured all demonstration payloads were designed to be harmless and easily reversible.

The educational nature of the project was always our priority. We designed the device to teach security concepts rather than to enable malicious activity. The payload examples we included demonstrate vulnerabilities without causing damage, allowing students to learn about the threats without putting systems at risk.

We also included discussions about defensive measures in our project presentations. Understanding attacks is only valuable if it leads to better defense. We talked about mitigation strategies like disabling USB ports in BIOS, using USB port blockers, implementing device whitelisting, and enforcing physical security policies.

XIV. Conclusion

We successfully built a working multi-mode HID attack tool using the STM32F303RET6 microcontroller. The device works as intended - it can pretend to be a USB keyboard and run different demonstration attacks through an easy-to-use LCD menu. This project proves that we can create advanced security tools using affordable embedded systems hardware that is readily available.

This project has good educational value in multiple ways. Students get practical experience with USB communication, embedded programming, and real-time systems. The project also shows concrete examples of cybersecurity vulnerabilities that exist in everyday devices. This helps students understand security threats that might otherwise be difficult to grasp just from reading about them.

From a security perspective, this work shows why physical security is important. Many people focus on network security, software bugs, and encryption, but physical access attacks like HID injection are serious threats that can bypass many traditional security measures. Understanding these vulnerabilities helps us develop better protection methods and security policies for organizations.

The hands-on nature of this project made abstract security concepts tangible and memorable. During our demonstrations, we observed that people who used the device or watched it in action gained a much deeper appreciation for USB security risks than those who only read about such attacks. The immediate visual feedback of watching the device automatically execute commands on a computer created a lasting impression about the importance of physical security.

Our project also demonstrates the versatility of modern microcontrollers. The STM32F303RET6 proved capable of handling complex USB protocols, managing a user interface, and executing time-critical operations simultaneously. This shows that embedded systems can be used for sophisticated applications beyond simple sensor reading or motor control. The skills we developed in this project are applicable to many other embedded systems projects in industrial automation, IoT devices, and consumer electronics.

The collaborative nature of our team effort taught us valuable lessons about project management and teamwork. Dividing responsibilities among eight team members required clear communication, regular coordination meetings, and well-defined interfaces between modules. We learned to use version control systems effectively, document our code properly, and integrate our individual contributions into a cohesive whole. These soft skills are just as important as the technical knowledge we gained.

In the future, we could add wireless connectivity to trigger payloads remotely, include more payloads for different operating systems, and integrate with professional penetration testing tools. The modular design we used makes it easy to add these features later while keeping the basic educational and demonstration functions we built. We could also develop a companion mobile app that provides more detailed control and monitoring of the device.

Another potential enhancement would be adding logging capabilities to record which payloads were executed, when they ran, and what the results were. This would be valuable for security training sessions where instructors want to review what students tested and analyze the outcomes. The logs could be stored on an SD card or transmitted wirelessly to a monitoring system.

We could also expand the project to include defensive demonstrations. For example, the device could show how system policies like disabled USB ports or device whitelisting prevent HID attacks. This would provide a more complete educational experience showing both offensive and defensive security techniques.

A. Learning Outcomes

Through this project, each team member gained valuable technical and professional skills. We learn how USB protocols work at a detailed level, including enumeration sequences, descriptor structures, and HID report formats. We developed proficiency in embedded C programming, real-time system design, and hardware-software integration. We also improved our debugging skills, learning to use tools like logic analyzers and protocol analyzers to diagnose problems.

Beyond technical skills, we learned about responsible disclosure and ethical considerations in cybersecurity research. We discussed how security researchers must balance the demonstration of vulnerabilities with preventing misuse of their work. This project taught us to think carefully about the implications of our work and to prioritize education and defense over potentially harmful applications.

B. Impact and Applications

The practical applications of this project extend beyond academic learning. Security professionals can use similar tools for authorized penetration testing to identify vulnerabilities in their organizations. The device serves as a conversation starter about physical security policies, helping organizations understand why they need to implement controls like locked server rooms, supervised visitor access, and USB port management.

Educational institutions can use this project as a teaching aid in cybersecurity courses. Provides a hands-on demonstration that makes abstract concepts concrete for students. The project could be replicated by other student groups as a learning exercise, with our documentation serving as a guide for implementation.

The project also raises awareness of the trust relationships built into modern computing. Many users do not realize that their computer automatically trusts any device that claims to be a keyboard or mouse. This project helps people understand that physical security is not just about preventing laptop theft but also about preventing unauthorized device connections that could compromise systems within seconds.

Acknowledgment

The authors express sincere gratitude to Shiv Nadar University for providing the resources and facilities necessary for this project. Special thanks to Prof. Rohit Singh and Prof. Sonal Singhal from the Department of Electrical Engineering for their invaluable guidance, technical mentorship, and continuous support throughout the development process.

We acknowledge the university's embedded systems laboratory for providing access to development tools, testing equipment, and workspace. We also thank our peers in the Electrical Engineering department for their feedback, suggestions, and collaborative spirit during the project development phases.

The authors are grateful for the funding and hardware support provided by the department, which allowed us to purchase the necessary components and development boards. Access to reference materials, academic journals, and online resources through the university library was instrumental in our background research and technical implementation.

Special appreciation goes to lab technicians and support staff who assisted with equipment setup, troubleshooting, and maintaining a productive work environment. Their patience and willingness to help at all hours, especially during critical testing phases, made a significant difference in our progress.

Finally, we acknowledge the broader open-source community and online forums that provided valuable insights, code examples, and troubleshooting assistance. The wealth of shared knowledge in the embedded systems and cybersecurity communities was an invaluable resource throughout this project.

Appendix
Bill of Materials

TABLE II
Bill of Materials

Component	Specification	Quantity
Microcontroller	STM32F303RET6	1
LCD Display	16x2 Character LCD	1
I²C Module	PCF8574 I/O Expander	1
Push Buttons	Tactile switches	4
Resistors	10kΩ pull-up	4
USB Cable	Micro-USB Type B	1
PCB/Breadboard	Prototyping board	1
Connecting Wires	Jumper wires	As needed

REFERENCES

[1] USB Implementers Forum, “Device Class Definition for Human Interface Devices (HID),” Version 1.11, USB.org, 2001.

[2] K. Nohl and J. Lell, “BadUSB - On Accessories that Turn Evil,” Black Hat USA, Las Vegas, NV, USA, 2014.

[3] STMicroelectronics, “STM32F303xB/C/D/E Reference Manual,” RM0316, STMicroelectronics, 2015.

[4] Hak5, “USB Rubber Ducky: Keystroke Injection Tool,” Available: https://shop.hak5.org/products/usb-rubber-ducky

[5] T. Neugschwandtner, A. Beitler, and A. Kurmus, “A Transparent Defense Against USB Eavesdropping Attacks,” in Proc. 9th European Workshop on System Security, 2016, pp. 1-6.

[6] J. Yiu, “The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors,” 3rd ed. Oxford, UK: Newnes, 2013.