Encrypted Wi-Fi Keyboard

Andrew Lu and Qi Guo, CMPSC 176B, Winter 2020

Introduction

The goal of this project is to create a wireless keyboard capable of securely communicating keystrokes over long distances.

Most keyboards on the market today communicate with devices through either a physical connection (e.g. USB and PS/2) or through Bluetooth / Bluetooth Low Energy (BLE). However, both methods place a limit on how far the keyboard can be from the connected computer, as physical connections are limited by cable length and Bluetooth connections can only transmit up to 100 meters, depending on the class used. This project aims to circumvent this limitation by instead using the user's internet connection to communicate keystrokes via a client-server socket communication application.

Implementation *(Changes to the original project proposal have been bolded)*

The Wi-Fi keyboard will feature a wired keyboard connected to a Raspberry Pi Zero W that runs a client application that listens for keystrokes on the attached keyboard. If keystrokes are detected, the client (keyboard) will encrypt them using public-key encryption to ensure safety of information. The keystrokes will then be sent via Wi-Fi to a computer running the server application, which will then decrypt the message and apply the decrypted keystrokes on the server computer. Messages will be sent using either a TCP or UDP socket connection, to be determined after a performance analysis is done comparing both methods.

We chose to use C++ instead of Python to implement the keyboard (client side) application because we were unsuccessful in getting the keyloggers we had originally researched to perform as expected. We have instead opted to build our own keylogger by reading and decoding the raw USB input from the keyboard. To accomplish this, we made a mapping between the key name and the HID scan code representation.

We will still use Python to implement the computer (server) side of the project. We started out with TCP connection, and we will test out UDP connection later. Python was selected for its ability to perform low-level socket programming natively using its libraries, and the many keystroke emulator packages available. Python was also selected for its ability to execute shell commands, which may be useful for this project.

Keystroke emulation will be done using the Win32 API, since we use Windows computers. We will be using PyWin32, which is a Python wrapper for the Win32 APIs. Eventually, we would like to switch to a keystroke emulator that works on multiple platforms.

RSA public-key cryptography will be added to our application using the PyCryptoDome package in Python.

Timeline

Milestone 1 (Week 1):

- Develop client and server applications capable of sending messages using both TCP and UDP sockets. **Done**
- Perform a performance evaluation comparing the two methods. **In progress**

Milestone 2 (Weeks 2-3):

- Develop a client application capable of recording keystrokes from a Linux machine **Done**
- Develop a server application capable of taking those keystrokes and applying them on a Windows machine. **In progress**
- If time permits, see if we can make the client and server applications work on all major platforms (Windows, Mac, Linux).

Milestone 3 (Week 4):

- Merge the two client applications and the two server applications together to allow us to send keystrokes unencrypted over a network. **In progress**
- Perform a performance evaluation comparing the two communication protocols, along with Bluetooth using an existing Bluetooth keyboard. **To do**

Milestone 4 (Weeks 5-6):

- Implement public-key encryption on the client and server applications, allowing data to be transmitted in an encrypted state. **To do**
- Perform a final performance evaluation comparing encrypted versus non-encrypted messages on both TCP and UDP sockets. **To do**

Milestone 5 (Weeks 7-8, but mostly if time permits):

- Instead of manually entering the IP address of the server into the client to establish the connection, use broadcast messages to allow the client to automatically discover the server, given that both devices are on the same network.

Given the short timespan of this class, I envision that I will be able to at least complete up to Milestone 4 by the end of the course. As a result, these will be the expected deliverables for this project:

- Client application capable of logging keystrokes from a keyboard and sending them over a socket connection
- Server application capable of receiving client messages and applying keystrokes on the server computer
- Messages exchanged are encrypted using public key (asymmetric) encryption

What's Been Done

- Qi Guo is now a member of this project group. We are now a team of two.
- We tested the Python and Linux keylogger programs, but both did not work with the Raspberry Pi, so we made our own using C++.
    - To do this, we researched the specification for USB HID keyboard input, and wrote a keylogger program to decode the input.
    - We were successful in printing out keyboard input into the terminal.
- We used C++ to implement the keyboard side to send messages.
    - The C++ script will initiate a TCP connection with the Python server using the message specification listed below.
- We used Python to implement the computer side to receive message.
    - The Python script accepts the incoming TCP connection and prints out the message sent by the keyboard side.
    - Keyboard emulation will come later once we finish both the TCP and UDP implementation of the client and server programs.
- We have defined the following specification for message packets:
    - Every packet is two bytes long
    - The first byte contains a number, between 0 – 255, that represents the HID scan code of the key pressed
    - The second byte contains the action, i.e. pressed (1), held (2), or released (0).

What's Left

- We will create a version of our program that uses UDP instead of TCP and perform a performance evaluation comparing the two.
- We will complete the server-side Python script to emulate the received keystrokes.
- After the above actions are completed, we will add encryption to our messages.
- If time permits, we will try to implement automatic device discovery.

Performance Evaluation

To evaluate the performance of our Wi-Fi keyboard, we can measure the latency and the success rate of input keystrokes at a fixed distance from a host computer for each of the various methods of communication. We hope to be able to compare the following five methods of wireless communication:

- TCP, without encryption
- TCP, with public-key encryption
- UDP, without encryption
- UDP, with public-key encryption
- Bluetooth (using a standard Bluetooth keyboard)

The latency for the socket methods will be computed by subtracting the time of the initial keystroke on the client computer from the time the keystroke is applied on the server computer.

The success rate will be computed by applying a fixed number of keystrokes on the client computer and measuring the number of keystrokes that were applied on the server computer. For the socket methods, this test will be conducted in varying environments, each with a different Wi-Fi connection quality.

References

Socket Programming using Python

- Basic Tutorial: https://www.geeksforgeeks.org/socket-programming-python/
- Socket Library Documentation: https://docs.python.org/3/library/socket.html

Custom Keylogger

- USB Scan Codes: https://www.win.tue.nl/~aeb/linux/kbd/scancodes-14.html
- HID Usage Tables: https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf
- Shell Commands in C: https://linux.die.net/man/3/system

Keyboard Emulator

- PyWin32 for Windows: https://pypi.org/project/pywin32/
- SendKeys Command (to apply keystrokes): https://ss64.com/vb/sendkeys.html
- Shell Commands in Python: https://janakiev.com/blog/python-shell-commands/

Public Key Encryption

- How it Works: https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/
- Encryption using RSA in Python: https://medium.com/@ashiqgiga07/asymmetric-cryptography-with-python-5eed86772731