Man-in-the-Middle Attack on Bluetooth Devices

Objective

The objective of this project is to mount a Man-In-The-Middle (MITM) attack to intercept and relay communication between a Bluetooth-enabled host device and a Bluetooth-enabled keyboard.

Introduction

To forward the keystrokes from a Bluetooth keyboard to a host, both devices must support the Human Interface Device (HID) Profile. There are two important features that are associated with this profile: Host-Device Virtual Cabling and Simple Secure Pairing (SSP).

Virtual cable is used to emulate the experience provided by a physical cable between a device and a host. According to Bluetooth HID profile specification, the following two behaviors of virtual cable have direct impact on this project:

- A Bluetooth device should not establish a Bluetooth HID connection with more than one Bluetooth HID host
- A Bluetooth device should not allow Bluetooth HID connections from hosts to which it is not virtually cabled unless the device is in Discoverable Mode

Although a device can support multiple virtual cables, the bottom line is, the device cannot have a connection with more than one host at a time. On the one hand, virtual cable stops the MITM attack by disallowing eavesdropping; on the other hand, it helps the attacker by preventing the user from connecting to the keyboard when the attacker connects to it first.

The purpose of Bluetooth SSP is to protect against eavesdropping. SSP protects the user from MITM attack, which is a form of active eavesdropping, by using two association models referred to as numerical comparison and passkey entry.

The association model is selected based on the I/O capabilities of the two devices. The numeric comparison association model is designed for the scenario where both devices are capable of displaying a six digit number and both are capable of having the user enter yes or no. This model is usually activated for paring cell phones or PCs. The passkey entry association model is designed for the situation where one device has input capacity but isn’t capable of displaying six digits, while the other device has output capabilities. A typical example is pairing a PC and a keyboard.

In this project, we demonstrate an attack vector which leverages the virtual cabling concept to overcome the MITM protections, provided by the SSP protocol and intercept the keystrokes from a keyboard to the target user’s host device. In our experiments, we used an Android phone as the host device, a Ubuntu laptop for the attacker and assume that there is only one Bluetooth keyboard in the environment. Also, it is assumed that these devices are in communication range of each other and the attacker is aware of the target user’s Bluetooth address. The overall attack vector can be divided into three phases: Discovery Phase, Impersonation Phase and Pairing Phase. In the discovery phase, the attacker's device scans the environment for a Bluetooth keyboard and connects to it as soon as one is found. Once the attacker connects the keyboard, the keyboard is no more discoverable. Next, the attacker moves to the impersonation phase and impersonates his device as a keyboard. When it’s done, the laptop will appear as the only keyboard listed on user's phone. As only one keyboard is visible to the user, he initiates a connection to it. Upon receiving a connection request from the victim, the attacker begins the pairing phase and enforces passkey-entry association model to complete the pairing process. The implementation details of each of these phases are described in the following sections.

Discovery Phase

In a typical scenario, when a user wishes to use the Bluetooth keyboard as an input device for his phone, the user presses the connect button on the keyboard and puts it in discoverable and connectable mode. This is followed by searching for the keyboard device in the user's device. Upon discovering a keyboard, the user connects to it. In order to ensure that the attacker connects to the keyboard quicker than the user, we wrote a Bluetooth program which constantly scans the environment for discoverable devices. When a device receives an inquiry request from a remote device, it replies with its Bluetooth address and its class. The class of a device (CoD) is a 24 bit value which indicates its type (such as computer, phone, and peripheral) and services it provides (such as networking, object transfer). Although, CoD of keyboards vary depending on the additional services it provides, the last 12 bits of the device must be 0x540. For each discovered device, the program checks if the device is a keyboard-type device and connects to when found.

To connect to the remote keyboard, we re-used some existing modules (available in Bluez library) to connect to a HID. We built an independent executable which connects to the remote keyboard and bypasses the pairing process completely. As mentioned earlier, although HID profile mandates security for a host-keyboard connection, such constraints are not placed for other HIDs such as joysticks, mouse. Hence, the existing HID module has been developed with security as an option, to connect to any HID. We exploit this flexibility and create an insecure communication between the attacker's laptop and the keyboard. This customized HID module creates two L2CAP connections with the remote device; one for control channel (PSM 17) and another interrupt channel (PSM 19) to communicate the keystrokes.

Impersonation Phase

Once a HID device is connected to a host, a host-device virtual cable is created the device begins serving the host. The HID device, now, transitions to a non-discoverable mode and stops scanning for inquiry packets. As the keyboard is connected to the laptop, it becomes invisible to the user's phone. At this stage, the Bluetooth program, renames the laptop with the keyboard's user-friendly name and changes its CoD to 0x000540. The attacker's laptop will appear as a keyboard to a remote device and thereby, successfully completing the impersonation phase. It must be noted that altering the name and CoD of the laptop does not affect its communication with the legitimate keyboard.

Pairing Phase

As the attacker’s device is only visible as keyboard to the user’s phone, the user would initiate a connection to this device. The Bluetooth daemon running in the attacker’s device handles such incoming connection request and automatically negotiates to use Numeric-Comparison association model for authentication as both the devices support Input and Output capability.

The goal for the attacker, at this stage, is to force Bluetooth daemon to use passkey entry association model. This is possible because even though the attacker cannot control how Bluetooth daemon responds to paring requests, he can manipulate I/O capability, if he initiates the connection. To achieve this goal without rebuilding BlueZ, the attacker listens to an incoming pairing request from the target user. As soon as the attacker receives such request, he blocks the user’s device and immediately sends out a pairing request to the user. In attacker’s request, the I/O capability is set to “KeyboardOnly” (constant value of 0x02) and as a result, the two devices will use passkey entry association model for authentication. To implement the same, we used an agent program available in BlueZ test utilities which is used to initiate a pairing process in command line. We changed the I/O capability parameter of this program to “KeyboardOnly” and obtained the desired effect.

This solution leads to two problems. First of all, when the attacker blocks user’s request, the user will be prompted with an error message that says “Unable to pair with BT Keyboard”. The error message is inevitable. Ideally, the more trivial and innocent an error message looks, the more likely the user will choose to ignore it. Among 60 errors defined in Bluetooth specification, “ACL connection already exist error”, (error code 0x0b) and “Unspecified error” (error code 0x1f) are good candidates. But initiating them properly is very difficult and so the attacker will have to allow for such notifications and take his chances.

Secondly, when user sees the passkey entry request, he will type the key on the legitimate keyboard, meanwhile, the attacker, as the man in the middle, must forward the keystrokes to user’s phone. To forward the keystrokes from legitimate keyboard to the victim’s device, we used an open source implementation of HID client. This HID client program forwards input events (such as keystrokes) that occur on a locally attached input device to a remote device via Bluetooth.

The MITM is successfully set up at this point.