Coercive Man-in-the-Middle

Ashwin Nambiar
3rd Year, Department of ECE
Amrita Vishwa Vidhyapeetham
ashwin0nambiar@gmail.com

Season Cherian
2nd year, Department of CSE
Amrita Vishwa Vidhyapeetham
seasoncherian@gmail.com

Hariprasad K V
2nd year, Department of CSE
Amrita Vishwa Vidhyapeetham
kannankv78@gmail.com

Ashwin Kumar
2nd year, Department of CSE
Amrita Vishwa Vidhyapeetham
ashwinkumarcr07@gmail.com

***Abstract*—This report presents a method to sniff data transferred between a successful Bluetooth Low Energy Connected devices without the knowledge of either of the devices. We can do this with the help of a Bluetooth hardware platform (Ubertooth in our case), a platform to perform Man-in-the-Middle attacks such as Gattacker and a Master-Slave device duo. We used a Linux system and a Raspberry Pi to fulfill those roles.**

Man-in-the-Middle attacks is an attack that has got a high success rate when it comes to Bluetooth devices. But if a BLE connection has been successfully established, information leakage is improbable. So this report discusses the sniffing data from an already established connection. We also discuss some future prospects in this subject.

I. Bluetooth

A. A Brief Overview

Bluetooth Low Energy hit the market in 2011 as Bluetooth 4.0. The development of BLE was a major development in the field of Internet-of-Things (IoT) devices. Tough there are some overlaps with the Bluetooth Classic, it is quite different from it in terms of its functionality. While Bluetooth Classic is a comparatively faster means of data transfer continuously so ideally useful for streaming music or receiving calls in your car stereo and stuff like that. BLE differs from classic in the fact that though not as fast, it consumes considerably low power which makes it ideal for small devices which only transfer data of much smaller magnitude less periodically. This can help conserve battery, particularly for wearable devices which make use of BLE for data transfer with its environment. Thus BLE has been used vastly in IoT devices due to its considerably low power consumptions and also the fact that it "sleeps" when it is not transferring data adds to its advantages.

B. Working of BLE

Bluetooth works in the same spectrum range as Classic ie 2.4 GHz. A BLE network comprises mainly of two components: the central device, mostly a smartphone or PCs having more CPU processing power; and a peripheral device, which is usually a sensor or any other low energy device of that sort, connected to the central device. The devices advertises two types of data: Advertising packets and Scan-Response packets; of which Advertising packets are mandatory. Both the payloads are similar, both able to contain 31 bit of data but Advertising packets needs to be constantly sent to inform the central devices in range of its presence.

The GAP or Generic Address Profile of a device determines the role of the particular device in the network and thus controls the advertising and connections. It makes the device visible and controls it's interaction with the world. GATT or Generic Attribute Profile controls the data transaction between the Central and Peripheral devices. GATT comes into play only after the connection is established and advertising is stopped. It will stop advertising when a connection is established ensuring only one device is connected at a time. GATT makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table.

II. Technical Background

Here we give a background info on how BLE connections work and the possible challenges that might be encountered. Since we are focused on attacks after the connections we don't do an in-depth analysis of device advertising.

A. BLE Connection

The most important thing about Bluetooth connections is that, the connection is exclusive, that is at a time only one central device can connect to a peripheral device. So once a connection os established, it stops sending advertisements thus it remains largely invisible when it is connected.

Upon successfully establishing a connection, communication happens both ways, ie from both the central device and the peripheral device. GATT defines the low level communication between the devices and deals with how data is transfered between the devices. It defines a hierarchy on the transferred data by organizing the data into sections called services which are in turn further divided into sets of user data called characteristics. Attribute value hold the data content of the specific attribute which can be of any data type thought the length is restricted to 512 bytes.

1) Services: The topmost tier which categorizes conceptually related attributes into a single category inside the GATT. As mentioned earlier services are again categorized into characteristics. They are naturally 16 bit UUID which are basically categorized into two: Primary and Secondary services. While the primary services is the standard GATT service which

Preprint to be submitted

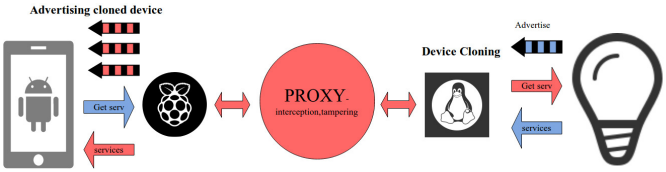

Fig. 1. A concise representation of a MitM attack

contain relevant useful functionalities of the GATT server, secondary services functions as a modifier having no existence on its own, thus they rarely used. A GATT can contain more than one service declarations in it.

2) Characteristics: The lowest level in the GATT transaction hierarchy which are the containers of user data. Similar to the services, characteristics distinguishes via 16 bit UUIDs. The characteristics have two attributes: the characteristic declaration, which contains the meta data of the user data; and the characteristic value, which contains the user data. This is the part of the GATT which concerns with the interaction with the BLE peripheral, both to and fro.

B. Data Sniffing Emulation and Capture

Our attack takes place only after a successful BLE connection is established thus advertisements are obviously out of the picture as the device stops sending the moment a successful connection is established. Our idea is to emulate the central device and take its place instead of the original device to get control of the peripheral as well as the thus having access to otherwise inaccessible information. For that we should first break the connection by emitting a RF using SDR which should match the base frequency of the Bluetooth chip to break the connection, emulate the device, sniff data packets from the Data transfer channels covertly. there also lies the challenge of successfully emulating the device with their MAC addresses, specific UUIDs and characteristics. Upon successfully emulating the packets, we use a packet capture like Wireshark to capture the transfered packets and read them.

C. Introduction

After a connection is established there is only data transfer and broadcasting between the 2 devices along 40 channels, 2 MHz each. 37 channels are used for connection data while ch: 37, 38 and 39 are used for broadcasting. In a connection there is only one broadcaster while the other is the observer. BLE connections use a technique known as frequency hopping spread spectrum wherein the broadcasting channels are changed from time to time.

III. Man-in-the-Middle attacks

When the devices are in connected state a frequency hopping algorithm is used to cycle through the data channels.

$$fn + 1 = (fn + hop)\%37 \quad (1)$$

But since the broadcasting is done only through 3 channels we can round up the probabilities to these three channels for sniffing useful data and it won't "hop" any further. So we will be focusing on sniffing data from these 3 channels. We make use of Ubertooth, a capable Bluetooth sniffer which also has an open source software platform and also an SDR to block connection paths along with a Raspberry Pi as an operating platform.

There are a couple of open source BLE sniffing packages and platforms to perform Man-in-the-Middle attacks like Gattacker, BLE Suite and BTLEJuice available which are all effective for our attack. The goal of the attack is to manipulate the information sent between the broadcaster and the observer, without their knowledge.

A. Setup

The main components include an Ubertooth, to capture and transmit data packets; an SDR to transmit RF of specific frequency to create interference in the connection a Linux system and also a Raspberry Pi. In this setup the Raspberry Pi becomes the Master device while the Linux system becomes the Slave device. After scanning the type of chip set being used in the IoT device we are able to find the frequency range of data transmission, thus we can emit RF of the required frequency to break the connection. We then have the Ubertooth to capture the packets that are being advertised and use it to emulate the device using the Slave device. The Master device on the other hand starts sending advertisements in large magnitudes to ensure that the devices connects with the Slave device.

B. RF Transmission and Emulation

As mentioned before a connection is established between the smart bulb and the mobile device, we scan the devices to find the hardware being used which can give us the technical details like the frequency of its transmission, base UUIDs and details like that. We then use the SDR to transmit a RF of the frequency which matches the discovered frequency of working. Thus we are able to break the already active connection thus the devices start their advertising again. This time, however, we make use of Gattacker to capture these advertised data packets and emulates device by mimicking the devices properties. Gattacker creates a platform to use captured packets from Ubertooth to emulate the device in the Raspberry Pi. It also successfully clones the MAC address of the bulb to spoof the Mobile phone, so as to match the GATT cache of the mobile device.

After the successful emulation, there are 3 devices advertising data packets at the same time. The bulb, the mobile and the emulated Linux Slave device. But there is a difference in the magnitude of the transmission between the bulb and the Slave device. Since both are trying to connect to the device, there should be a factor which would determine mobile device connect with the Slave instead of the bulb itself. Gattacker also enables us to advertise packets in much larger magnitude compared to the advertisements of the bulb. This is because, bulbs transmit advertisements in lesser magnitudes to minimize power consumption and thus the large amount

of advertisements almost guarantees the establishment of a successful connection between the slave device and the bulb.

C. Packet Capture

1) Important parameters:

a) Bluetooth Address: A unique 48 bit address that is commonly known as `BD_ADDR`. The first 24 bits identifies the manufacturer while the lower 24 bits are unique to that address. It is quite similar to the MAC Address in IP connections

*b) **Access address**:* Upon the establishment of a successful connection, we have access to the data that are being transferred between the connected devices. As mentioned before there are 40 channels of data transmisson, where 3 channels ae used for broadcasting. The frequency hopping spread spectrum makes interference difficult but not impossible. Here, they use a parameter called the Access Address which ensures that the data channels have negligible chances of collision while data transfer occurs. In fact, this address acts as the identification for a connection. It is 4 bytes long which is used for the master device access address identification.

*c) **Index**:* The channel of transmission ie the frequency index of the channel used.

*d) **Protocol Data Unit or PDU**:* The PDU contains info on the data that is being transferred that is the type of data transferred, the `SCAN_REQ` a request given by the scanner device to start transmission and `SCAN_RES`, a response given by the Central device upon receiving the `SCAN_REQ`.

IV. Future Plans

A. Advancement in SDR

Software-defined Radio (SDR) is a programmable transceiver with the capability of operating various wireless communication protocols without the need to change or update the hardware. SDR is a technology for radio communication. This technology is based on software-defined wireless protocols, as opposed to hardware-based solutions. This translates to supporting various features and functionalities, such as updating and upgrading through reprogramming, without the need to replace the hardware on which they are implemented. This opens the doors to the possibility of realizing multi-band and multi-functional wireless devices.

Due to shared nature of wireless communication we can easily monitored communication between two devices and emits false message to block communication. Nowadays increased use of software defined radio (SDR) technology makes any types of jammer device using same hardware with little modification in software. A jammer transmits radio signal to block legitimate communication either overlapping signal with more power or reducing signal to noise ratio. Existing jammer detection methods like packet delivery ratio (PDR), packet send ratio (PSR), bad packet ratio (BPR) and signal to noise ratio (SNR) can effectively detects jammer, here we have proposed novel method for jammer detection using communication parameter used in SDR like synchronization indicator, iteration and adaptive signal to jammer plus noise ratio (ASNJR). This system uses that parameter which is readily available in system so computation has been reduced and ASNJR also has been adaptively updated with and without presence of jammer.

Conclusion

Bluetooth security has always been a topic of discussion considering its vast application in IoT devices. Through our project we convey that through an already established BLE connection, though difficult, it is not impossible to sniff data packets and thus has a vulnerability that can be exploited.

Acknowledgment

The team would like to thank all the members of our CTF team bi0s for their unrelenting support lend to this endeavor. We also acknowledge Vipin Pavithran, our mentor and Arvind S Raj for their support, assistance and also for the equipment provided for our project.

References

[1] Gattacker by Slawomir Jasek (https://github.com/securing/gattacker)
[2] Ubertooth Wiki (https://github.com/greatscottgadgets/ubertooth/wiki)
[3] Bluetooth SIG Documentation (https://www.bluetooth.com/specifications/bluetooth-core-specification)
[4] Uri Shaked (https://medium.com/@urish/even-more-bluetooth-smart-bulb-hacking-fe888a1ab601)
[5] BlueZ Documentation (https://git.kernel.org/cgit/bluetooth/bluez.git/tree/doc)