Detection and Prevention of data transfer through Bluetooth by unauthorized devices on Android OS 8,9 & 10

MSc Research Project
MSc Cyber Security

Nash John
Student ID: 21154341

School of Computing
National College of Ireland

Supervisor: Dr. Vanessa Ayala-Rivera

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nash Jacob John		
Student ID:	21154341		
Program:	MSc Cyber Security	**Year:**	2022-2023
Module:	MSc Research Project		
Supervisor:	Dr. Vanessa Ayala-Rivera		
Submission Due Date:	15th December 2022		
Project Title:	Detection and Prevention of data transfer through Bluetooth by unauthorized devices on Android OS 8,9 & 10		
Word Count:	7201	**Page Count:**	17

Signature: Nash Jacob John

Date: 15th December 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Detection and Prevention of data transfer through Bluetooth by unauthorized devices on Android OS 8,9 &10

Nash Jacob John
21154341

Abstract

Bluetooth is a device that is short-range and could be used to transfer data between devices and can also be used to create a local area network with other users. In our present world, we use Bluetooth to connect to the Internet of Things devices, which helps in the communication and transfer of data with ease. Bluetooth is a widely adopted technology due to benefits such as it uses very less battery consumption to send and receive data between devices and there is a very extensive range for this device compared to the previous version. However, one of the main problems is that it is vulnerable to certain attacks. One of these is remote code execution (CVE-2020-0022). Bluetooth has two versions one is called the classic Bluetooth and another one is called the Bluetooth low energy. As Bluetooth technology is being used by most smartphone users, there tends to be a serious remote code execution security flaw in android versions 8, 9, and 10. On this specific android OS version, an attacker could get access to the android device's root privilege by sending improper packet data. In this paper, we focus on Bluetooth zero-click RCE – BlueFrag attack, this is a type of attack in which the attacker can gain a reverse shell on android OS without the user's permission, to prevent this attack we will be monitoring the packet and advertising signals sent and received by our device. This approach should help prevent the remote code execution attack on these devices. This security flaw was first discovered by the fuzzing technique using the tool called Frankenstein.

Keywords: Bluetooth, Bluetooth low energy, advertising, packet, GATT.

1 Introduction

Bluetooth technology was developed in 1994, and this technology has made a great impact on our day-to-day life. Since the creation of Bluetooth technology, there have been so many innovations in this technology. The Bluetooth classic was the first version that was introduced for wireless calling for the device. As time went by there is a new version of Bluetooth called Bluetooth low energy. This feature was an introduction to the Bluetooth 4.0 version.

According to CVE-2020-0022, it was reported an attacker was able to access the root privileges of the android phone without the knowledge of the user. One of the reasons why Bluetooth is capable of this is because Bluetooth usually receives a connection request from a nearby unauthenticated device. And the packet received by Bluetooth is processed by the Bluetooth chip called a controller, The controller then sends this data to the application layer. The latest android OS such as android 11, 12, and so on has prevented this attack. But smartphones that are running Android OS 8, 9, and 10 still have this vulnerability. According to the android community there exist thousands of users running this OS version. We will be addressing the vulnerability and finding developing a prototype in which the packets will be filtered so that the connection could safely authenticate between devices and blacklist all unauthorized Bluetooth devices.

2 Related Work

Bluetooth, which is now a necessary component of daily life. The majority of our IoT gadgets are connected utilizing Bluetooth to communicate with one another via our smartphones. There has been an uptick in smartphone sales worldwide since the pandemic touched our life. Android runs most of the sold devices. In recent years, there has been a noticeable increase in the number of smart technology goods available, including Bluetooth-enabled wireless headphones, smart watches, smart TVs, and smart appliances. One important fact about which we are all ignorant is Bluetooth's implementation of security. Because we all have faith in the manufacturers, we also don't worry about those things. However, Bluetooth technology is not as safe as we might imagine. Attacks on Bluetooth in the recent past have resulted in reverse shelling of the victim's smartphone. We could comprehend the structure of Bluetooth to do additional research on this subject.

2.1 Background on how Bluetooth works.

One of the first technologies we use is Bluetooth, which has been around since the invention of the phone. For mobile phone users, Bluetooth was largely employed to provide hands-free calling. Bluetooth is a short-range wireless technology that enables data transmission and reception between two devices. Ultrahigh-frequency (UHF) radio waves in the industrial, scientific, and medical (ISM) short-range radio frequency band are used by the system, which functions using a frequency-hopping spread spectrum. The data is sent by packets to the 79 assigned Bluetooth channels. According to the Bluetooth Wiki, Bluetooth is a packet-based protocol with a main/follower architecture in which the main may connect to as many as seven followers in a piconet [Wiki Bluetooth]. In today's most advanced smartphone devices, Bluetooth Low Energy and Bluetooth Classic have different battery requirements (Dian, Yousefi, and Lim, 2018). Current Bluetooth 5.2 uses this idea, which was modified from Bluetooth 4.0. One of the levels that makes up the Bluetooth component is the Transport layer, which we use in our network-connected devices to send and receive network packets. The layer for Bluetooth is comparable, as can be seen in Fig. 1 (Wiecha et al., 2016).

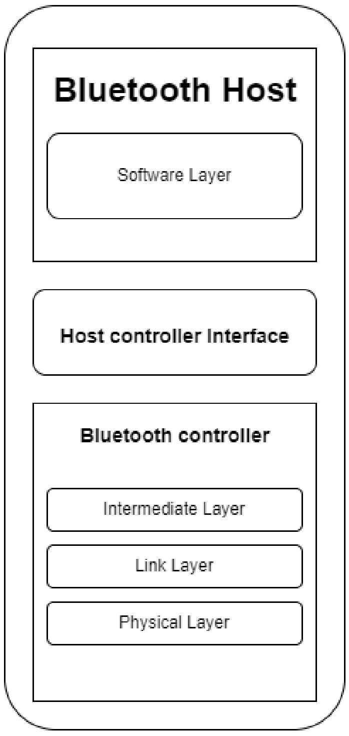


Figure 1: Bluetooth BLE System Architecture

In Fig. 1 above, we can observe that the Bluetooth controller receives data from other devices, processes it, and then sends it to the Host Controller Interface (HCI), which provides fundamental register/data operations to the application layer for user interpretation. (Wiecha et al., 2016).

2.2 Detecting the vulnerabilities in Bluetooth

Approximately 900 million devices have Bluetooth chipsets on the market today. BLE's primary function is to transmit and receive data between devices and sensors. Many attacks, including firmware attacks and man-in-the-middle attacks, have been recorded. According to the research, Bluetooth 4.0 inhibits assaults of this nature by generating LTK and encrypting data using elliptic-curve Diffie-Hellman. Turning off the Bluetooth device when not in use, updating the firmware to the most recent version, and resetting the Bluetooth by switching to airplane mode, which clears the Bluetooth's memory, are some of the common defenses against these attacks. This was the author's intended strategy (Pallavi and Narayanan, 2019). Some cases are more serious than this, such as those when the fix is not appropriate. One such attack is the so-called "blue borne attack," which uses Bluetooth to spread across the air and from one device to another. The attacker can take complete control of the target device and see the user in real-time if the user is infected with the blue-borne virus. The author of this paper discusses how to secure Bluetooth and how to find most of its security flaws. One typical method of carrying out the attack is to provide the user with the proper packets, which are then saved in the target's memory. The attacker then uses this packet to execute the code which will result in privilege escalation. The author has developed a BAPF framework to counteract this kind of attack, which will filter out dangerous packets and clear the memory after a Bluetooth connection is severed. This framework will install a firewall to assist in identifying malicious transmissions (Almiani et al., 2019).
By enabling Bluetooth's IDS system to accurately recognize commands with a 99.9% accuracy, the author of this study proposes a technique for detecting intrusion. The author

also makes recommendations for how to thwart the most recent CVE attacks, which are known to employ AI and machine learning. Some of the typical traits of this attack are battery drains and increased power utilization, which causes low battery. With the help of the whole data set, the author trained the model so that the user could anticipate the attack (Bluetooth Anomaly Based Intrusion Detection System - ProQuest, 2017). The IDS system, however, is unable to detect a different assault on this system known as the blur tooth. The Cross-transport key derivation CTKD is used in this attack. We can exploit both Bluetooth and Bluetooth Low Energy (BLE) by focusing on only one of the two, making our assaults the first examples of cross-transport Bluetooth exploitation. The employment of CTKD as a weapon in attacks is also new in these. Without a trustworthy attacker model, the attacks can still be conducted. They can be utilized immediately on any device, for instance (including those that offer BT and BLE SC, and SSP with a strong association). Due to the blurring of the lines between BT and BLE security, we refer to our assaults as "BLUR attacks." The usual protections against this attack involve turning off Bluetooth while not in use, upgrading the firmware with the manufacturer, and using secure pairing with other devices (Antonioli et al., 2022).
In the paper that follows, we investigate how to go around the Android 8 smartphone's Bluetooth 5.2 safe pairing process. The BLE programming framework is not given instructions on how to do SCO mode operation, and the BLE specification does not mandate that the initiator must operate in this mode. The BLE programming framework of the initiator could have serious flaws that could be exploited to carry out downgrade attacks, making the BLE pairing protocols run insecurely without the user's knowledge. These flaws include the handling of SCO initiation, status management, error handling, and bond management. The following tenets serve as the foundation for Android-based devices. The same kind of victim devices can be obtained by an attacker to examine the applications and communication protocols. The attacker is unable to physically grasp the phone. Our attacks use Bluetooth weaknesses as opposed to many others in the past without requiring the target device to have any malicious software installed (Zhang et al., 2020).

A secure association mechanism, such as Passkey Entry or Numerical Comparison, was formerly utilized to link the Android mobile and its peer device. This assumption places attackers in a more plausible and challenging situation. The fact that all of the tactics described in this study can be applied even if Just Works or pairing has not been established between the Android and its peer device should be stressed. Android. We employ the SCO mode as a programmable option to make the BLE programming framework compatible and give apps the ability to defend themselves from attackers. whether the Android mobile device's BLE setting corresponds with the most recent Support for older devices in the BLE standard (Gentner, Günther, and Kindt, 2022).

Bluetooth Low Energy (BLE) is being used by an increasing number of Internet of Things (IoT) devices to communicate in an energy-efficient way. Since the author did not disclose any security measures, it is clear from the previous study that they will be able to mimic other devices and access the victim's phone. We have included the remedy for addressing that issue in this document. By 2023, BLE will enable up to 5 billion devices, making it one of the most popular protocols for IoT devices (such as smart locks, lighting, or thermostats) (Träskbäck, 2018). Due to hazards that permit unauthorized device access, user fingerprinting, and inference of sensitive information on the device, worries regarding BLE's security are growing because of its widespread use. Although numerous solutions have been put forth to reduce some risks, most of them concentrate on protecting the BLE device alone, neglecting the end-to-end connection between a BLE device and a user device in this instance (Hassan et al., 2018).

The authors developed BlueShield, a monitoring solution that is legacy-friendly, vulnerability-agnostic, and non-intrusive, to address these issues. 23rd International Symposium on Research in Attacks, Intrusions, and Defenses by the USENIX Association 397 A monitoring system called BlueShield may identify spoofing attempts in a fixed BLE network and alert the user. It carefully looks for irregularities in the distinctive fusion of cyber and physical components of advertising packets defining the identity of the BLE device to find the effects of spoofing attempts. Since a successful spoofing attack requires the attacker to broadcast the fake advertising packets before the target user's arrival, such acts reveal the attacker in advance and provide BlueShield plenty of time to detect before any real harm is done to the victim or the user (Wu et al., 2018).

2.3 Vulnerabilities in L2Cap

One of the most widely used short-range wireless communication technologies in recent years is Bluetooth technology. One of the many applications for the Bluetooth standard is the Synchronous Connection Oriented (SCO) and Connection-Less Asynchronous (CLA) connectivity (ACL). The term "SCO link" refers to a system of recurrent reservations and real-time service transfers (mainly voice application). To convey asynchronous data, ACL links are designed for use. High-quality data application traffic, such as music and/or video file transfers, etc. Only SCO Link and ACL Link are supported; Logical Link is not. Only ACL is supported by the Logical Link Control and Adaptation Protocol (L2CAP). Since the Bluetooth Radio and Baseband layers serve as the cornerstone for the packet transport in the L2CAP layer, it is essential to understand their characteristics and use earlier research findings as a guide for subsequent L2CAP layer studies. L2CAP's main responsibility is to keep lower-layer transport protocol features hidden from higher-layer protocol stack levels. The basic, retransmission, and flow control modes are supported by the L2CAP layer. We additionally assume that there is only one active slave during the transmission of the L2CAP Protocol Data Unit to further simplify the analysis and modeling (PDU) (Hua) (PDU)(Hua and Zou, 2008).

The Service Discovery Protocol (SDP) enables network devices, applications, and services to discover and seek out complementary network devices, applications, and services that are required for properly completing tasks. A wide range of unique service discovery protocols have been warmly received by the network computing industry, and these technologies have been eagerly adopted. In this paper, the Bluetooth SDP protocol stack—which uses special service discovery algorithms—will be discussed. Sun improved Java by introducing Jini to it. Microsoft promotes the adoption of Universal Plug and Play (UPnP). The service location protocol is officially acknowledged as official by the IETF (SLP). Additionally, Salutation, a non-proprietary service discovery protocol that is presently used by various shipping solutions, has been promoted. Salutation is a coalition of independent, top technology businesses (Avancha, Joshi, and Finin, 2002).

The SDP has been the victim of several assaults that prevent users from understanding when and how the attacker got access to the platform, as was explained in this paper when we first learned about Bluetooth. As a result, the accompanying page lists several popular mitigation strategies, including Default settings that ought to be modified to meet ideal standards. ensuring that the machinery is contained and operates within a safe range. Devices are set to their lowest power setting for this. Use lengthy, random PINs to reduce the risk of brute-force assaults. frequently changing the device's default PIN (i.e., once every two months). Except for when pairing is required, devices are by default set to the inaccessible mode. Devices must be in discoverable mode for the majority of existing discovery technologies to detect them. When not in use or when it is not necessary, Bluetooth on a device should be turned off, especially when in a public setting like a bar, club, or other meeting place. By setting this, users can prevent seeing adverts from other Bluejackers. refusing to input passkeys or PINs when unexpectedly prompted. Update your software and drivers frequently to ensure you have access to the latest product updates and security fixes. Users are advised not to utilize unsupported or insecure Bluetooth-enabled devices or modules. Versions 1.0 through 1.2 of Bluetooth are included in this.

Whenever necessary, device pairing. Users should never forget that any pairing should happen in a private, secure environment. If this is done, pairing communications will be less susceptible to being intercepted by enemies. (2018) (Lonzetta and others). Most of the common faults that the Bluetooth L2Cap layer oversaw were understandable to us based on the publications. As well as software updates that detect and prohibit unfamiliar devices through real-time monitoring, phone manufacturers like Google and Apple have helped to limit most of these well-known assaults. Most unknown vulnerabilities, on the other hand, cannot be found traditionally. We could study these new vulnerabilities using the innovative L2Cap fuzzing method.

2.4 Using Fuzzing techniques to find vulnerabilities in Bluetooth

To ensure the foundation of Bluetooth device confidence, L2CAP, the lowest layer used by Bluetooth services, was chosen for our analysis. Core field modification and state guidance are L2FUZZ's two key techniques. With the use of state guidance, L2FUZZ can match each L2CAP state with the proper instructions based on its events, functions, and actions. To modify a state or determine whether an assault is successful against a certain state, the mapped commands can be used in this situation. The core fields of L2CAP packets, which determine the port and channel, are modified solely by L2FUZZ using a core field mutation technique. By modifying only, the core fields and leaving the other components alone, L2FUZZ can give more reliable findings. L2CAP is a basic Bluetooth protocol since every Bluetooth application needs a connection between the master and slave devices. To use Bluetooth applications, the master must be aware of the service ports and channels of the slave services, which are controlled by L2CAP. Suppose we wish to use Bluetooth file transmission in this situation. The master and slave devices initially exchange an encrypted key during this procedure via the controller stack. The sharing of service ports and channels is then done using the L2CAP layer. Since these ports and channels, they construct Radio Frequency Communications (RFCOMM) and Object Exchange (OBEX) links that make use of file transfer applications. The payload of an L2CAP message is composed of Code, Identifier, Data Length, and Data Fields.

The phrases Code and Identifier, respectively, stand for the packet ID and the instruction code. In Bluetooth 5.2, there are 26 L2CAP commands, each with a different set of Data Fields, which decide the Data Fields. For instance, two Data Fields are included in the "L2CAP connection request": Source Channel ID and Protocol/Service Multiplexer (PSM, port number) (SCID). SCID, Result, Status, and Destination Channel ID are the four data fields that make up the "L2CAP connection response" (also known as DCID). L2FUZZ was generally successful in locating vulnerabilities, but it was unable to perform long-term fuzzing; when a fatal flaw is triggered on the target device, it forcibly disables Bluetooth.

The tester must manually reset the device before conducting another test (Lacava et al., 2022). To fix this issue, we'll consider employing a virtual setting. Although L2FUZZ can look through the target's response packets to look for flaws, the root cause cannot be found at once. We intend to address this problem by considering internal log hooking that investigates the crash root cause, such as ToothPicker. Since Bluetooth devices are closed-source, black boxes, it is difficult to determine code coverage. We emphasized that

Frankenstein was capable of measuring code coverage in a constrained way using binary, even though it required difficult tasks like firmware emulation. Finally, despite being able to access the majority of L2CAP states, there are still some circumstances in which it is not able to. There may be restrictions on the states that the target device can enter, for instance, when L2FUZZ (as a master) interacts with a slave target device. We're considering using tactics like inserting programs to control the test target's state transitions. (Park *et al.*, 2022).

Although we will cover a very dependable method in which we might find vulnerabilities utilizing Frankenstein in the present study, we can see that there are several limits to employing the L2Fuzzer. Before connecting, we can achieve Bluetooth zero-click RCE by painstakingly fuzzing the parts of the Broadcom firmware that are accessible. Both stacks are confused and vulnerable when utilized in the same ways, even though they have diverged since Cypress purchased some of Broadcom's Bluetooth code in 2016. Emulation and fuzzing reveal undocumented proprietary firmware. The tool was created in C, a programming language that interacts with the firmware image and enables the testing of firmware hypotheses and the identification of relevant code pathways. (Hou, Zhang, and Man, 2020, p. 0).

Android-based Frankenstein usage The BlueFrag RCE is one of three flaws that were found. By creating a physical device snapshot with Frankenstein and then simulating it with Quick Emulator, the entire stack is fuzzed (QEMU). The emulated firmware connects to a real Linux host, leverages task and thread switches to fuzz various handlers, and uses a virtual modem to give over-the-air data. Without any additional modifications, it uses QEMU in user mode. It is feasible to incorporate and transfer snapshots of the actual hardware's operational state to the simulated environment. In the sections that follow, we present a capability demonstration of our technique utilizing the CYW20735 Bluetooth controller. But additional firmware is also supported. It is possible to connect to the emulated virtual Bluetooth chip using even a sophisticated operating system like Linux and other operating systems that enable UART Bluetooth, such as macOS. The procedure for reinstalling the CYW20735 firmware is described in detail below (Ruge *et al.*, 2020).

2.5 Summary

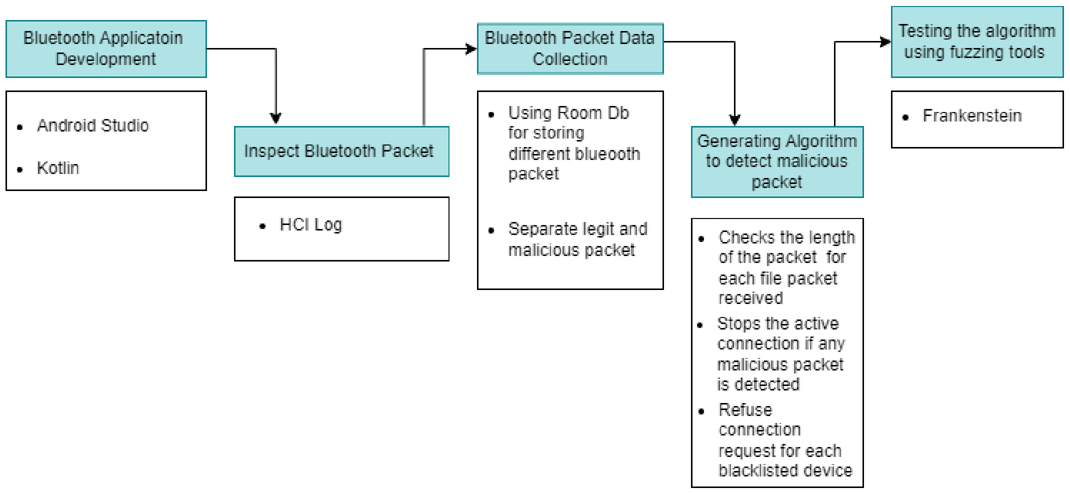
We can observe from the publications that Bluetooth classic and Bluetooth low energy are mostly responsible for vulnerabilities. Most businesses have recently updated their software to protect against typical attacks like Man-in-the-Middle, Bluetooth spoofing, eavesdropping, BlueBorne assault, etc. We learned from the (Ruge et al., 2020) research that the BlueFrag vulnerability, which is still present in Android devices running Android OS 8, 9, and 10, is well known. Nearly 80% of users still use Android OS, and inadequate heap sanitization and L2CAP packet monitoring are two major reasons why an attacker would be able to escalate their privileges. When we look at all the devices running this version of Android, we can see that this is still the case. I suggest figuring out a way to filter out packets that prohibit remote code execution via packets that are being received by Bluetooth through the L2Cap layer for android devices.

3 Research Methodology

Our aim for the proposal is to identify and segregate bad actors while establishing a connection with other devices. To achieve this, we will be creating an application for android devices that will be compatible with all android OS. In the below sections, we will be looking into the various steps in which we will be implementing the prototype.

3.1 Project Implementation

To implement our project, it is essential to know the terminology that we will be required to learn and understand to develop this project. In the below diagram we could see how we are going to implement our project:



Bluetooth Applicatoin Development
Android Studio
Kotlin
Inspect Bluetooth Packet
HCI Log
Bluetooth Packet Data Collection
Using Room Db for storing different blueooth packet
Separate legit and malicious packet
Generating Algorithm to detect malicious packet
Checks the length of the packet for each file packet received
Stops the active connection if any malicious packet is detected
Refuse connection request for each blacklisted device
Testing the algorithm using fuzzing tools
Frankenstein

Figure 2: Project Implementation diagram

Bluetooth Application development: First the project implementation is to develop and Bluetooth application. This application should be able to connect with other devices and should be able to send and receive data between them. To develop this application, we will be using android studio. Android studio is a platform in which android applies applications created. This software is open source and could be installed and used by any user second part is selecting the programming language. The android application could be developed in both languages, that are java and kotlin. Kotlin is a new language and the official android developers recommend developing apps in kotlin. So, we will be using kotlin as our official language in developing this prototype.

Inspecting Bluetooth packet: After developing the application our next aim is to get the Bluetooth packets from the android OS. To achieve this, we will have to go to the user's phone settings and enable **Developer options,** inside the developer option we will then select the HCI snoop log. After that, we will have to turn off and turn on the Bluetooth so that the system will generate an HCI log data called the **hci_snoop.log**. In the newer version of the smartphone the manufacturers not allowing users to access this data, to access hci_snoop.log we will have to root our phone. This doesn't concern us because we will be using the Nordic semiconductors **Android-BLE-Library**, this library will help us access all the packets that are being sent between devices and will work on most android devices.

Bluetooth packet Data Collection: Once we can get the hci_snoop.log data from the android device storage, we will be importing the log file into our application and read all the log data inside it. Once read the data will be stored inside our database, which is called the Room DB which is an SQLite database, provided by the android developers. If we are not able to get the log file from the memory, it won't make any impact on our application since we will be monitoring and storing real-time packets sent via Bluetooth devices, and will help us get a better understanding of different packet structures.

Generating algorithm to detect malicious packets: The next stage of our project is to evaluate the packets collected from our database. From the series of packets collected will be able to determine whether each packet is malicious or not from the length of the packet and by which is received by our device. From each packet, we will also be able to determine which device had sent us a particular file, so that in the future will add those devices to the blacklist and if that device tries to communicate with our device, the device will automatically close the connection before evening paring with it.

Testing algorithm using the fuzzing tool: To test whether our prototype is working properly as intended, we will be using a tool called the Frankenstein. This is a fuzzing tool used to generate a malicious packet and send it to a Bluetooth receiver. This tool was used to detect the remote code execution on an android device. Thus, this tool will help our project to get better testing results.

The packets broadcast and received by the Bluetooth layer must be evaluated for the proposed question to produce a successful conclusion. L2Cap is also known as the Logic Link Control and Adaptation Layer Protocol. Peer-to-peer connections in Bluetooth are made possible by this layer. Just above the Host Control Interface is the L2Cap (HCI). Both synchronous and asynchronous packet transfers are handled by HCI. Data transport uses asynchronous packet (ACL) transfers, while audio or hand-free profiles use synchronous packet (SCO) transfers.

The connection between the host and the controller is handled by HCI. The host in this instance is an Android smartphone (Samsung Galaxy S10). Further examination reveals that we were able to gather HCI logs for every android smartphone that was being collected. This might be produced by manually triggering the HCI snoop log from the Developers option or by programmatically activating our Bluetooth application. The HCI log is often kept on our phone's internal memory, and the most popular location to save it is at /data/misc/Bluetooth/logs.

After successfully creating a log on our device, we can now use our app to monitor all of the logs in real-time by utilizing a content provider to access the information from the phone's internal memory. A typical packet that Bluetooth sends and receives during a connection is shown below. Typically, a hacker will attempt to overflow this kind of packet, which will cause Bluetooth to malfunction. This can cause the application to crash or result in reverse shelling. Use of the malicious packet-sending tool Frankenstein might be used to carry out this misbehavior.

4 Design Specification

The main objective of our application is to stop remote code execution on Android devices via Bluetooth. To achieve this goal, we will be developing an algorithm in which the packets will be examined before it being sent to the application layer. One of the main reasons why the attacker can get root privileges is by sending the improper packet to Bluetooth. This improper packet mainly exceeds the length of the prescribed Bluetooth packet and hence the Bluetooth couldn't process the data properly which leads to root access. In this proposal, we will be developing an algorithm that will evaluate every packet which is being received by Bluetooth before sending it to the application layer. Thus, we could prevent the malicious packet from accessing the Bluetooth and could help us to eliminate the connection with that system. The below diagram shows how the algorithm will be working when a packet is received by Bluetooth.

4.1 GATT Server

Two Bluetooth Low Energy devices exchange data utilizing the ideas known as Services and Characteristics according to the Generic Attribute Profile, or GATT for short. It uses the Attribute Standard (ATT), a general-purpose data standard, to store Services, Characteristics, and associated data in a straightforward lookup table with 16-bit IDs for each item. Once a dedicated connection has been made between two devices, GATT comes into play, proving that the GAP-mandated advertising procedure is complete. The most crucial GATT principle to remember is that connections are exclusive. As a result, only one active connection between a BLE peripheral and a central device (such as a phone) is permitted at any given moment. Until the present connection is severed, a peripheral device that has connected to a central device c[1]eases promoting itself and is no longer reachable by or accessible by other devices.

4.1.1 GATT Transaction

The phone or tablet is referred to as the GATT Client, and the peripheral GATT Server, which is where the ATT lookup information and descriptions of services and features are maintained. The first device, the GATT Client, initiates each transaction. The secondary device, the GATT Server, is contacted for a response. A central device (the GATT Client) and a peripheral device exchange data, and each transaction are started by the primary device, as depicted in Fig. 3.

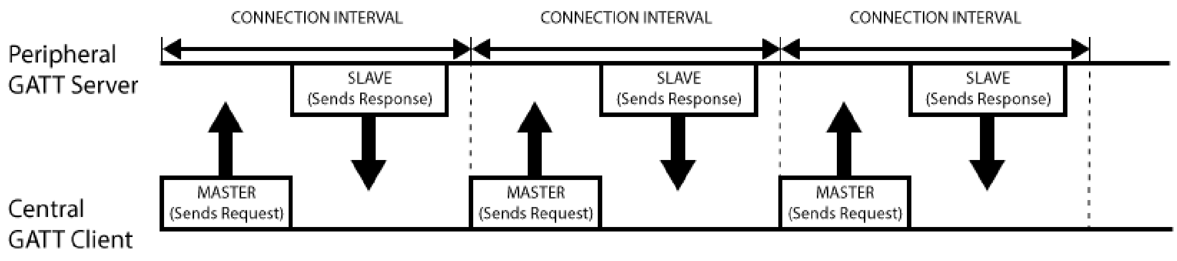


Figure 3: GATT Server and client communication

4.2 Services

Services have types of data known as characteristics that are used to divide data into logical entities. Each service utilizes a UUID, which can be either 16 bits (for officially adopted BLE Services) or 128 bits (for unauthorized BLE Services), to differentiate itself from other services. The service may include one or more features.

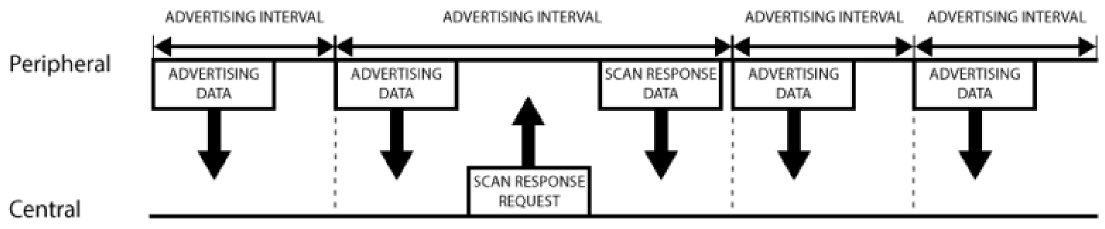
[1] *Bluetooth Technology Overview* (2014) *Bluetooth® Technology Website*. Available at: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/.

4.3 Characteristics

The Characteristic, which includes a single data point but may also include a lot of accompanying data, such as the X, Y, and Z values from a 3-axis accelerometer, is the lowest-level concept in GATT transactions. Each Characteristic uses a fixed 16-bit or 128-bit UUID to identify itself, just like Services. Using the standard characteristics released by the Bluetooth SIG (which ensure compatibility between BLE-enabled HW/SW) or announcing our unique bespoke features, which are only understood by your peripheral and SW, are the two options available to us.

4.4 Advertising

In the 2.4GHz ISM band, there are 40 physical channels for Bluetooth low energy, each spaced apart by 2MHz. Data and advertisement transmissions are the two transmission kinds that Bluetooth defines. Thus, 3 of these 40 channels are devoted to advertising, while the other 37 are for toa.
All peripheral devices start, at least initially, in advertising mode, whether it be a beacon (transmitting position, weather, or other data) or a fitness watch establishing a long-term connection with a host (tablet or phone). Devices can broadcast information describing their goals through advertising.



[2]**Figure 4: How advertising works diagram**

[2] *Introduction to Bluetooth Low Energy* (2014) *Adafruit Learning System*. Available at: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

4.5 Algorithm

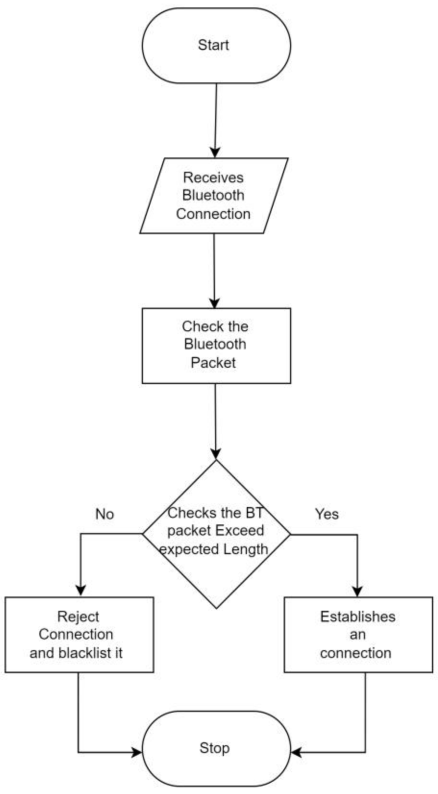

Start
Receives Bluetooth Connection
Check the Bluetooth Packet
Checks the BT packet Exceed expected Length
No
Yes
Reject Connection and blacklist it
Establishes an connection
Stop

Figure 5: Bluetooth Packet filtering algorithm

From Fig. 4 we could we a visual representation of how the packet filtering will work. When the Bluetooth pairs with another Bluetooth device, and starts the transfer of data between the device, the algorithm will start monitoring each packet being received. The algorithm checks the packet data length before sending it to the application layer and if it doesn't satisfy the predefined conditions the application stops the transfer of data between the devices and disconnects the Bluetooth connection and blacklists the device.

5 Implementation

We need to set up a lab in the Android environment to have the proposal solved perfectly. This entails developing an Android application that will display all of a Bluetooth device's features. We must become familiar with the programming language that was used to create the project to accomplish this. Learning the essential programming abilities to create this Android application is the first stage in the project's implementation.

The next step is to design a clean user interface for the application. There should be two pages minimum in the application. The available Bluetooth devices will be displayed on one screen, and the connected devices and their specifications should appear on the second page. The prebuilt XML language found inside the Android Studio might be used to accomplish this. The third step entails designing the project's architecture. The main goal is to make all application layers transparent to detect faults faster and to make the source code more readable to make it more secure against malicious assaults that we were unable to foresee during the project's development phase. The fourth step is to install the necessary dependencies for the project so that we may include numerous prebuilt classes in our lab, which will facilitate the development of the application. The fifth stage is to create a Bluetooth interactive application that will enable us to keep track of the packets being sent through the various Bluetooth layers. The prebuild class, which is offered by the Android library, may accomplish this. Additionally, to watch the log on our Android smartphone, we must enable the HCI snoop log. The sixth stage is to extensively analyze the Bluetooth packets being sent and received, and we will be creating a regex by the ACL and SCO packet format.

5.1.1 Platform Selection

The BlueFrag vulnerability on Android smartphones running Android OS 8, 9, and 10 allowed for remote code execution. which is still widely used by individuals worldwide. According to the most recent poll conducted by Android Studio (a platform for developing Android applications), Android OS 8 is currently installed on 86.7% of handsets, followed by Android OS 9 at 75.1% and Android OS 10 at 58.9%. We can see from the percentages that developers are still concentrating on Android OS 8 when creating their applications to reach as many consumers as possible. Therefore, the android studio will be a good option for fixing the problem for the suggested problem statement.

5.1.2 Language Selection

Two languages are supported by Android Studio while creating applications. Java and Kotlin are the two languages that are supported. Since the inception of Android applications, Java programming language has been used to create them. Kotlin was introduced in May 2019 as an official language for creating Android applications. After the Kotlin language was introduced, all applications were converted from Java to Kotlin. Kotlin employs less boilerplate code than Java, which makes it simpler for developers to read the code when compared to Java. This is one of Kotlin's key advantages over Java. The programming language's lower level of vulnerability, when compared to Java, is another benefit of the Kotlin dialect.

5.1.3 Architecture

We would need access to the Bluetooth HCI Log, which the Android smartphone will automatically generate and keep inside the local memory, to construct a Bluetooth application that could monitor Bluetooth packets that are being received from other devices in real time. We will need built-in Android features like a content provider to access this. The Android team created a “Content Provider” service that allows users to access local files and files used by our application. Considering this situation, the Android development team recommends the MVVM design as one of the finest architectures. The developer will be able to separate the various layers into individual classes and packages using this design.

5.1.4 UI Development

Android Studio offers the XML language for creating user-friendly applications with a clean user interface. Every project created using Android Studio incorporates XML, and the UI may be easily changed to meet the needs of the project. XML is fairly simple to comprehend and pick up. Therefore, for this project, XML will be a good option for creating the UI framework.

5.1.5 Approaching the problem statement

Now that the platform, architecture, and user interface (UI) components have been considered, we must develop Android applications that carry out Bluetooth-like operations. The program should display all nearby Bluetooth devices as well as all connected devices from the Bluetooth memory.

The next step is to retrieve the HCI Log from the current smartphone. There are two methods for us to get that. The first involves manually enabling the HCI snoop log by going to the developer's option in a smartphone's settings. A different strategy is to programmatically enable this choice after getting the user's approval. To enable this option programmatically, we should require Bluetooth Admin power. By including the “Bluetooth Admin” permission in the Android application manifest, this power may be attained.

After acquiring the smartphone's HCI log, we might be able to create our L2Cap secure connection with other devices (HCI Requirements | Android Open-Source Project, 2020). The Bluetooth hardware in our phone is now directly communicated with by the application we utilize. Delivering malformed packets to the Bluetooth receiver, which Bluetooth is not meant to handle, is one of the main ways the Bluefrag attack is carried out. To address this issue, we'll monitor each Bluetooth packet that is sent and received in our app's HCI and L2Cap layers. We might create a Regex that would filter each packet to prevent privilege escalation by an attacker to obtain reverse shelling or to check if each packet has any malicious payload that could prevent Bluetooth from functioning.

5.2 Technical Analysis

To effectively respond to the suggested question, we must evaluate the packets that the Bluetooth layer is sending and receiving. The Logic Link Control and Adaptation Layer Protocol is another name for L2Cap. This layer enables peer-to-peer connections in Bluetooth. The L2Cap is immediately above the Host Control Interface (HCI). HCI manages both synchronous and asynchronous packet transfers. Synchronous packet (SCO) transfers are used for data transfer, whereas asynchronous packet (ACL) transfers are used for audio or hand-free profiles.

HCI oversees managing the connection between the host and the controller. In this case, the host is an Android smartphone (Samsung Galaxy S10). Further investigation demonstrates that we can obtain HCI logs from every Android smartphone for every device being collected. This might be achieved manually using the Developers option's HCI snoop log or automatically using our Bluetooth application. The most frequently used location to save the HCI log on our phone's internal memory is at /data/misc/bluetooth/logs.

After establishing a log on our smartphone, we can now access all the logs through our app in real-time and use the content provider to extract the data from the phone's internal memory. The image below depicts a typical packet that a Bluetooth connection transmits and receives. This form of the packet is frequently overloaded by hackers, which causes Bluetooth to operate incorrectly. The application can crash as a result of this, or you might get reverse shelling. The cloud's Frankenstein tool's malicious packets could be utilized to carry out this bad deed.

6 Evaluation

In this section we will be analyzing our android application called the Packet Filter, from which we will be discussing our findings which are discussed below in detail:

6.1 Discovering Bluetooth devices

On advertising our Bluetooth application, we could observe that the Bluetooth beacon sends a request to each possible MAC address nearby. And the request from all the devices is being received by our application and could be monitored in our android Log. Of the 100 scanned devices there exist only 20 devices that could be connected through Bluetooth. In the application, we have set a condition in which the application will show only devices that could communicate through Bluetooth. From the 20 Bluetooth devices the names of those devices are not shared properly or are read “null” from this we could assume that those devices are vulnerable and could cause harm to the user.

6.2 Discovering Authenticated Bluetooth devices

After we have completed our initial filtering process, we will be displaying the devices which are connectable through Bluetooth and we will be getting information about the devices such as the MAC address, and data status which should that if we have shared data with the device during the advertising stage, the initial value of this variable is 0 and if the device had shared some data with another device then the value of this variable will change from 0 to higher value. This means that without the authorization of the user the Bluetooth has transmitted some data, and this notifies us that the Bluetooth device that we are interacting with is not secure and we could blacklist this device and could prevent it from making a future connection.

From the above findings, we could conclude that there are so many unauthenticated Bluetooth devices and one of them could be an attacker which could inject a malicious packet into the smartphone and thus gain the root privilege. This could be detected and prevented early on the android platforms with the help of this application.

7 Conclusion and Future Work

Bluetooth is getting more and more popular nowadays due to the extensive use of devices such as Bluetooth speakers, smartwatches, etc., and the convenience which this provides for connecting nearby devices and transferring data (audio, video, heartrate) makes it more usable by the users. The android operating system which is one of the most knowledgeable and user-friendly OS has almost 3 million users around the world. In this research paper, we have discussed how Bluetooth is working behind, different attacks that were performed on Bluetooth devices, and we have also learned the working of advertising which helps in acting as a beacon for other devices to discover and connect with our device. We have also learned the function of GATT and how they send and transfer data using services and characteristics.

Android OS is released every year with new security patches. But on older smartphones after a point in time, the vendors will stop pushing the latest OS releases. In the market of Android OS, if we take a percentage of OS which the users are using, we could see that Android OS 8,9, and 10 have more users than the latest OS version. The Bluetooth Zero-click RCE BlueFrag vulnerability has been patched in the latest version of android but still exists in the older version of android. These devices are still vulnerable to this attack and thus the data of the users could be compromised. In this research, we have developed an android application that could run on the older and newer versions of android OS and thus help the users detect the vulnerable device in the initial stage itself. This helps users stay protected while using the Bluetooth device indoors and outdoors. We have tried to develop an algorithm to detect the malicious packet being received by an external device and detect the data that has been received during the advertising stage. In future work, a machine learning algorithm could train and detect various attacks on Bluetooth. This could help us to learn and understand better the security side of Bluetooth and helps in making the users use a safer Bluetooth connection.

References

Almiani, M. *et al.* (2019) 'Bluetooth Application-Layer Packet-Filtering For Blueborne Attack Defending', in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 142–148. Available at: https://doi.org/10.1109/FMEC.2019.8795354.

Antonioli, D. *et al.* (2022) 'BLURtooth: Exploiting Cross-Transport Key Derivation in Bluetooth Classic and Bluetooth Low Energy', in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery (ASIA CCS '22), pp. 196–207. Available at: https://doi.org/10.1145/3488932.3523258.

Avancha, S., Joshi, A. and Finin, T. (2002) 'Enhanced service discovery in Bluetooth', *Computer*, 35(6), pp. 96–99. Available at: https://doi.org/10.1109/MC.2002.1009177.

Bluetooth Anomaly Based Intrusion Detection System - ProQuest (2018). Available at: https://www.proquest.com/openview/946b0be95c9089a0e779fe6cf855e0df/1?pq-origsite=gscholar&cbl=18750 (Accessed: 2 August 2022).

Dian, F.J., Yousefi, A. and Lim, S. (2018) 'A practical study on Bluetooth Low Energy (BLE) throughput', in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 768–771. Available at: https://doi.org/10.1109/IEMCON.2018.8614763.

Gentner, C., Günther, D. and Kindt, P.H. (2022) 'Identifying the BLE Advertising Channel for Reliable Distance Estimation on Smartphones', *IEEE Access*, 10, pp. 9563–9575. Available at: https://doi.org/10.1109/ACCESS.2022.3140803.

Hassan, S.S. *et al.* (2018) 'Security threats in Bluetooth technology', *Computers & Security*, 74, pp. 308–322. Available at: https://doi.org/10.1016/j.cose.2017.03.008.

HCI Requirements | Android Open Source Project (no date). Available at: https://source.android.com/devices/bluetooth/hci_requirements (Accessed: 2 August 2022).

Hou, D., Zhang, J. and Man, K.L. (2020) 'Enhancing the Security of Numeric Comparison Secure Simple Pairing in Bluetooth 5.0', in *2020 IEEE 19th International Conference on Trust, Security and*

Privacy in Computing and Communications (TrustCom), pp. 1622–1629. Available at: https://doi.org/10.1109/TrustCom50675.2020.00224.

Hua, Y. and Zou, Y. (2008) 'Analysis of the packet transferring in L2CAP layer of Bluetooth v2.x+EDR', in *2008 International Conference on Information and Automation*, pp. 753–758. Available at: https://doi.org/10.1109/ICINFA.2008.4608099.

Lacava, A. *et al.* (2022) 'Securing Bluetooth Low Energy networking: An overview of security procedures and threats', *Computer Networks*, 211, p. 108953. Available at: https://doi.org/10.1016/j.comnet.2022.108953.

Lonzetta, A.M. *et al.* (2018) 'Security Vulnerabilities in Bluetooth Technology as Used in IoT', *Journal of Sensor and Actuator Networks*, 7(3), p. 28. Available at: https://doi.org/10.3390/jsan7030028.

Pallavi, S. and Narayanan, V.A. (2019) 'An Overview of Practical Attacks on BLE Based IOT Devices and Their Security', in *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, pp. 694–698. Available at: https://doi.org/10.1109/ICACCS.2019.8728448.

Park, H. *et al.* (2022) 'L2Fuzz: Discovering Bluetooth L2CAP Vulnerabilities Using Stateful Fuzz Testing', in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 343–354. Available at: https://doi.org/10.1109/DSN53405.2022.00043.

Ruge, J. *et al.* (2020) 'Frankenstein: Advanced Wireless Fuzzing to Exploit New Bluetooth Escalation Targets', in, pp. 19–36. Available at: https://www.usenix.org/conference/usenixsecurity20/presentation/ruge (Accessed: 2 August 2022).

Träskbäck, M. (2000) 'Security of Bluetooth: An overview of Bluetooth Security'. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1357247eca49bedea0cb21439db85db3f2452384.

Wiecha, P. *et al.* (2016) 'Architecture and design of a Bluetooth Low Energy Controller', in *2016 MIXDES - 23rd International Conference Mixed Design of Integrated Circuits and Systems*, pp. 164–167. Available at: https://doi.org/10.1109/MIXDES.2016.7529724.

Wu, J. *et al.* (2018) 'BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks', p. 16. Available at: https://www.usenix.org/system/files/raid20-wu.pdf.

Zhang, Y. *et al.* (2020) 'Breaking Secure Pairing of Bluetooth Low Energy Using Downgrade Attacks', in, pp. 37–54. Available at: https://www.usenix.org/conference/usenixsecurity20/presentation/zhang-yue (Accessed: 2 August 2022).