Global Journal of Researches in Engineering: F
Electrical and Electronics Engineering
Volume 22 Issue 1 Version 1.0 Year 2022
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 2249-4596 & Print ISSN: 0975-5861

Bluelock a Tool to Prevent Bluetooth Attacks

By Gerardo Alberto Castang Montiel, Fernando Betancourt Duque & Luz Adriana Peña Salazar

Universidad Distrital Francisco José de Caldas

Abstract- The mobile device manufacturers and Internet Of Things are searching for more compatible and easy to connect protocols that increase coverage, which generates the user an effective experience when using different devices. Nevertheless, the constant updating process opens a security gap that exposes the users' personal information. Bluetooth is a communication protocol openly used by manufacturers because of their excellent information transfer capacities, allowing hackers to exploit it. The manufacturers work daily, generating security patches for communication protocols but this hasn't been enough to mitigate vulnerabilities. The security has always been on the manufacturer's side but the final user is limited to use a few security customization options to protect his network perimeter. Based on the wireless devices' background we wonder, Is there any way in which the user has the chance to reduce the possible dangers that wireless devices face? We can answer this question through the development of a research in the more vulnerable areas, simulating the attacks on a mobile device to generate a possible solution that allows the user to have more control over his bluetooth connections.

Keywords: *bluetooth, android, security, attacks.*

GJRE-F Classification: *FOR Code: 290903*

Strictly as per the compliance and regulations of:

Bluelock a Tool to Prevent Bluetooth Attacks

Gerardo Alberto Castang Montiel [α], Fernando Betancourt Duque [σ] & Luz Adriana Peña Salazar [ρ]

Abstract- The mobile device manufacturers and Internet Of Things are searching for more compatible and easy to connect protocols that increase coverage, which generates the user an effective experience when using different devices. Nevertheless, the constant updating process opens a security gap that exposes the users' personal information. Bluetooth is a communication protocol openly used by manufacturers because of their excellent information transfer capacities, allowing hackers to exploit it. The manufacturers work daily, generating security patches for communication protocols but this hasn't been enough to mitigate vulnerabilities. The security has always been on the manufacturer's side but the final user is limited to use a few security customization options to protect his network perimeter. Based on the wireless devices' background we wonder, Is there any way in which the user has the chance to reduce the possible dangers that wireless devices face? We can answer this question through the development of a research in the more vulnerable areas, simulating the attacks on a mobile device to generate a possible solution that allows the user to have more control over his bluetooth connections.

Keywords: bluetooth, android, security, attacks.

I. Introduction

Advances in wireless communication have faced compatibility issues, which has forced technology manufacturers to regulate communication protocols. One of the most popular has been Bluetooth used to interconnect mobile devices and IoT technology. This protocol was created in 1994 by the electric engineer Jaap Haartsen and it has been updated throughout the years that has allowed it to be one the best protocols due to its information transfer rate.

Because of its popularity, coming from manufacturers, this has been the target of hackers who have used vulnerabilities to attack other users and obtained their private information from their mobile devices. The response to these attacks is usually reactive. Such attacks have compromised thousands of users' information, even installing apps that act like malware, that allows access to the peripherals connected to the victim's device, like a camera, mic, GPS, and others.

Author α: Ingeniero electrónico, Universidad Distrital Francisco José de Caldas, Colombia. Correo electrónico: gerardocastang@gmail.com
Author σ: Tecnólogo en sistematización de datos, Universidad Distrital Francisco José de Caldas, Colombia.
Correo electrónico: duquefernandobetancourt@gmail.com
Author ρ: Tecnóloga en sistematización de datos, Universidad Distrital Francisco José de Caldas, Colombia.
Correo electrónico: luapenas@correo.udistrital.edu.co

Smartphones are the most used devices for Bluetooth technology due to its features and the storage capacity in Android. The manufacturers keep reactive during the attacks, generating patches and updates, saving the security recommendations.

To generate a preventive solution, we propose the user is able to manage the Bluetooth interface so it can be used without exposing private information when the device is active. To accomplish this, a testing environment is set up to deploy Bluetooth attacks to identify possible vulnerabilities that could be reduced by an Android app for Smartphones.

a) Glossary

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance and commercially sponsored by Google.

- *Attacks:* In this project an attack is a cyberattack, and it is the attempt caused by someone who wants to get control over an informatic system once he gets access to it.

The attacks have several purposes in order to cause damage through espionage to get money or to find vulnerabilities in the system.

- *BlueLock:* It is the created tool for Android devices that allows control over the Bluetooth interface by the user to manage connectivity among active devices.
- *Bluetooth:* It is a short-range wireless technology standard that is used for exchanging data between fixed and mobile devices over short distances using UHF radio waves in the ISM bands, from 2.402 GHz to 2.48 GHz, and building personal area networks (PANs).
- *Kali Linux:* Kali Linux *(formerly known as BackTrack Linux)* is an open-source, Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali Linux contains several hundred tools targeted towards various information security tasks, such as Penetration Testing, Security Research, Computer Forensics and Reverse Engineering. Kali Linux is a multi platform solution, accessible and freely available to information security professionals and hobbyists.
- *Pentesting and Ethical hacking:* The terms 'penetration testing' and 'ethical hacking' are often

used interchangeably when referring to internal cyber security tests, but they're not exactly the same. Penetration testing is a type of security test in which an organisation hires a certified professional to assess the strength of its cyber security defences. The goal of ethical hacking – like criminal hacking – is to find security vulnerabilities in an organisation's systems. However, as the word 'ethical' suggests, the person conducting the attack must have the organisation's approval before proceeding.

- *Information security:* The state of being protected against the unauthorized use of information, especially electronic data, or the measures taken to achieve this.
- *Vulnerability:* It is a failure in the security system in which the user can access a system to manipulate information, app or take control over the system.

II. Attack Deployment

Knowing the different types of attacks over Bluetooth, they are possible to duplicate through Pentesting. The tests done are known as "white box" (tandem), because we implement the attack environment using the platform pentesting Kali Linux, keeping track of every tool and designing the hacking tests for Bluetooth.

To begin, through Kali Linux we ensure that Bluetooth is activated using the following commands:

```
root@kali:~# service bluetooth start
root@kali:~# service bluetooth status
● bluetooth.service - Bluetooth service
   Loaded: loaded (/lib/systemd/system/bluetooth.service; enabled; vendor preset
   Active: active (running) since Sun 2018-11-11 11:35:52 -05; 5h 33min ago
           Docs: man:bluetoothd(8)
 Main PID: 622 (bluetoothd)
   Status: "Running"
           Tasks: 1 (limit: 4915)
   Memory: 4.0M
   CGroup: /system.slice/bluetooth.service
           └─622 /usr/lib/bluetooth/bluetoothd
```

After, we validate the Bluetooth interface (Mac Address)

```
root@kali:~# hciconfig
hci0:   Type: Primary  Bus: USB
    BD Address: C8:FF:28:A6:15:F1  ACL MTU: 820:8  SCO MTU: 255:16
    UP RUNNING PSCAN ISCAN
    RX bytes:3857 acl:0 sco:0 events:602 errors:0
    TX bytes:28371 acl:0 sco:0 commands:388 errors:0
```

We use the Bluelog tool from Kali Linux that allows us to see and register detectable devices on the Bluetooth network and it is executed as you will find below:

```
root@kali:~# bluelog
Bluelog (v1.1.2) by MS3FGX
-------------------------
Autodetecting device...OK
Opening output file: bluelog-2018-11-11-1655.log...OK
Writing PID file: /tmp/bluelog.pid...OK
Scan started at [11/11/18 16:55:26] on C8:FF:28:A6:15:F1.
Hit Ctrl+C to end scan.
```

Blue Scanner is another Kai Linux tool that allows us to scan through the Bluetooth network adapter, capturing connectivity features from connected devices on the network and it is executed as follows

```
root@kali:~# btscanner
```

Keyword "i" produces and scans all devices on the network and generates a file directory called by the MAC's device name with all the information obtained, using these tools we can start to validate the attacks and check their functionality.

- *Bluejacking attack:* To make this possible, we need to create a contact user, instead of the name, the message will be written, in the directory folder that will act as the attacker and the contact user is saved. After That, an area with several mobile devices is found and the option “send via Bluetooth" is chosen, and it is sent to the target devices.
- *Bluesmack:* a line executed directly from the attacker’s console device. which deploys several calls in a specific frame of time, causing the Bluetooth denial service in the host or hacked device. The script is:

```
root@kali:~# while read r; do l2ping -s 50 84:C7:EA:57:36:D7; done < numscans
```

we created a file called: numscans with a timer from 1 to n, to be used in the script and avoid memory leaking in the attacker's device.

```
root@kali:~# while read r; do l2ping -s 50 84:C7:EA:57:36:D7; done < numscans
Ping: 84:C7:EA:57:36:D7 from C8:FF:28:A6:15:F1 (data size 50) ...
0 bytes from 84:C7:EA:57:36:D7 id 0 time 5.91ms
0 bytes from 84:C7:EA:57:36:D7 id 46 time 63.44ms
Send failed: Connection reset by peer
Ping: 84:C7:EA:57:36:D7 from C8:FF:28:A6:15:F1 (data size 50) …
```

a) Attack results

We can see the Bluetooth protocol is designed to facilitate device connection, and it is a mandatory approach for IoT communication devices, so through the software we designed we do not want to limit or block the protocol’s functionality automatically by a service or app, but we want to give the user this option, designing an app that allows him to see the connectivity events, paired devices, and allows the user the chance to block any device at anytime.

III. Design Phase

a) Software structure

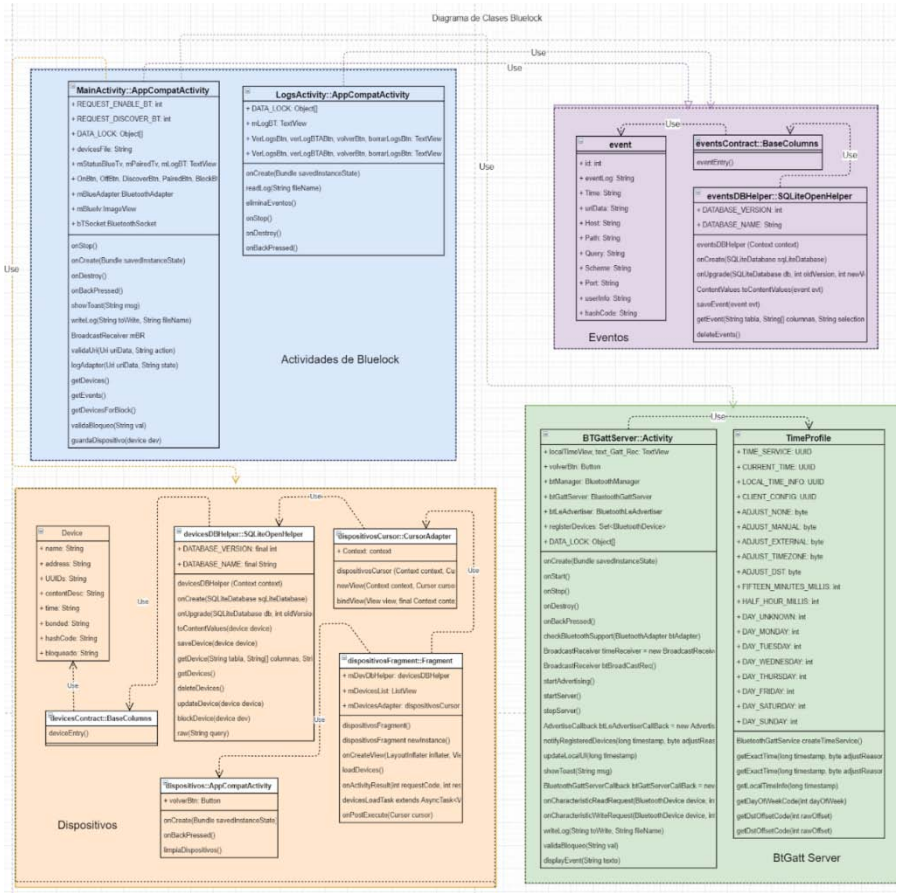

Fig. 1: Class Diagram

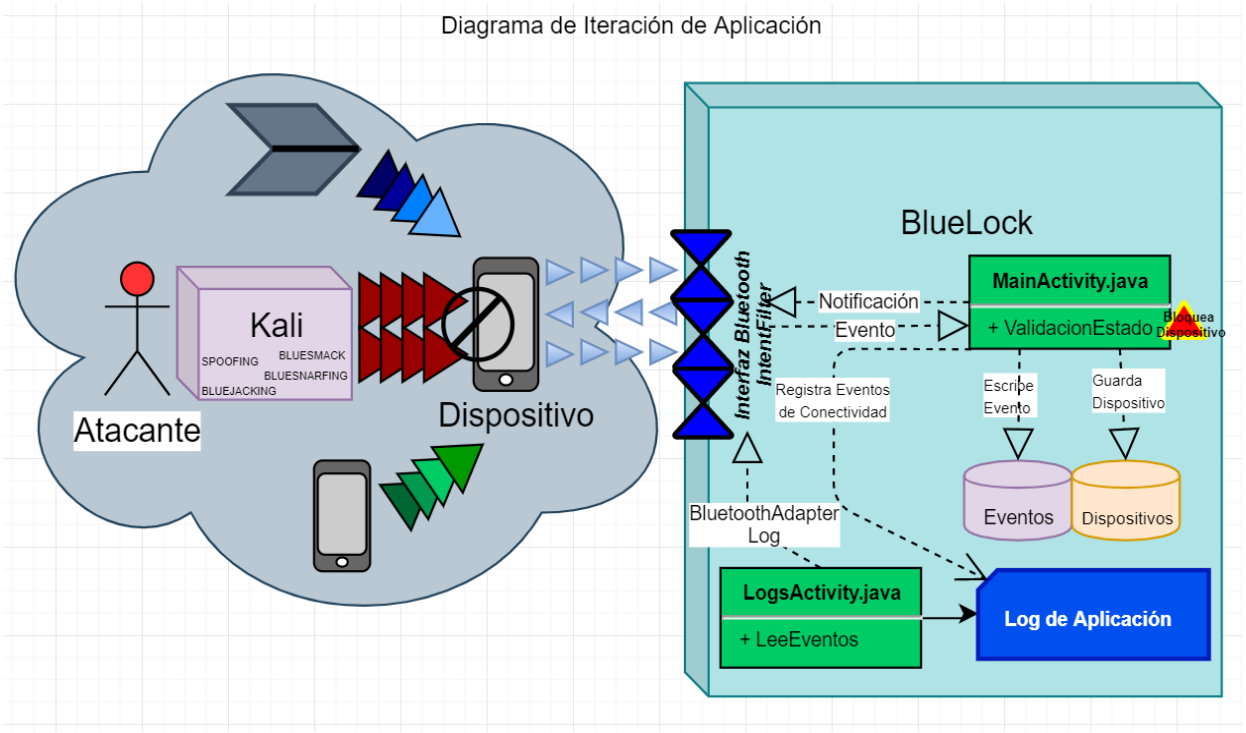

Fig. 2: Iteration Diagram

The diagram shows the device and the app running, participating in different communications with different devices, the device is in a susceptible environment when it is often attacked via Bluetooth. The device has a Bluetooth interface that communicates with the app (Bluelock) that manages the Bluetooth adapter.

The app (BlueLock) has an event filter that detects specific changes in the Bluetooth adapter, such as incoming connections, and registering all the information in a database, and the app log.

IV. Development Phase

App development

As a solution a mobile app for Android is proposed that allow people to control the Bluetooth interface communication, turn on and off the Bluetooth controller, enable finding new devices, consult paired devices, check the Bluetooth event's logs, block access to multiple devices.

App features

1. User's interface to control the Bluetooth device.
2. Generate bluetooth log events detected by the app.
3. block device interface through Bluetooth interface.
4. Generate a non-relational database that allows storing information about Bluetooth connected devices and their related events.

a) Code documentation

Bluetooth filter packages: The main feature consists in the BroadcastReceiver function, that allows sending and managing the communication events sent by the Bluetooth adapter in the Context.sendBroadcast. This function filters and controls states detected by the adaptor, does the actions generated by the bluetooth device and writes the log events.

```
//BroadcastReceiver
private final BroadcastReceiver mBR = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        String evento = "";
        boolean bloqueo = false;
        final Uri uriData = intent.getData();
        event evt = validaUri(uriData, action);
        eventsDBHelper dbHelper = new eventsDBHelper(getApplicationContext());
        String log = "", showToastLog = "";
        boolean logger;
        if(action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)
                ||action.equals(BluetoothAdapter.ACTION_DISCOVERY_STARTED)
                    ||action.equals(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)
                        ||action.equals(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)) {
            final int estado = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
BluetoothAdapter.ERROR);
            switch (estado) {
                case BluetoothAdapter.STATE_OFF:
                    evento = "<font color='blue'>BluetoothAdapter.STATE_OFF";
                    log += logAdapter(uriData,evento);
                    break;
                case BluetoothAdapter.STATE_ON:
                    evento = "<font color='blue'>BluetoothAdapter.STATE_ON";
                    log += logAdapter(uriData,evento);
                    break;
                case BluetoothAdapter.STATE_TURNING_OFF:
                    evento = "<font color='purple'>BluetoothAdapter.STATE_TURNING_OFF";
                    log += logAdapter(uriData,evento);
                    break;
                case BluetoothAdapter.STATE_TURNING_ON:
                    evento = "<font color='purple'>BluetoothAdapter.STATE_TURNING_ON";
                    log += logAdapter(uriData,evento);
                    break;
```

Fig. 3: Broadcast Receiver Function

Validation of the connection state of the device in the event "connecting". If the device is blocked, a remove Band event is generated which eliminates the paired device from the Bluetooth device list, blocking every attempt from other devices to connect.

```
case BluetoothAdapter.STATE_CONNECTING:
  evento = "<font color='purple'>BluetoothAdapter.STATE_CONNECTING";
  log += logAdapter(uriData,evento);
  BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
  bloqueo = validaBloqueo(device.getAddress());
  UUID uuid = UUID.randomUUID();
  if(bloqueo){
    try {
      Log.w("ACTION_ACL_CONNECTED","Entró a removeBond");
      Method m = device.getClass()
          .getMethod("removeBond", (Class[]) null);
      m.invoke(device, (Object[]) null);

      evento = "<font color='red'>Dispositivo Bloqueado ("+uuid+"):
"+device.getAddress()+"; "+device.getName();
```

```
            log += logAdapter(uriData,evento);
            showToast("Dispositivo bloqueado ("+uuid+") : "+device.getAddress()+" : "+
device.getName());
        } catch (Exception e) {
            Log.w("ACTION_ACL_CONNECTED", e.getMessage());
            evento = "<font color='red'>Error de Dispositivo Bloqueado ("+uuid+"):
"+device.getAddress()+"; "+device.getName();
        }
        log += logAdapter(uriData,evento);
    }
    break;
```

Fig. 4: Event Connecting

We validate the other BluetoothAdapter status. Every event generates a line in the Log events that can be consulted by the app. It was designed to have styles so when the .txt is rendered, shows with colors the severity of the case.

```
case BluetoothAdapter.STATE_CONNECTED:
            evento = "<font color='green'>BluetoothAdapter.STATE_CONNECTED";
            log += logAdapter(uriData,evento);
            break;
        case BluetoothAdapter.STATE_DISCONNECTING:
            evento = "<font color='green'>BluetoothAdapter.STATE_DISCONNECTING";
            log += logAdapter(uriData,evento);
            break;
        case BluetoothAdapter.STATE_DISCONNECTED:
            evento = "<font color='purple'>BluetoothAdapter.STATE_DISCONNECTED";
            log += logAdapter(uriData,evento);
            break;
        case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
            evento = "<font color='purple'>BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE";
            log += logAdapter(uriData,evento);
            break;
        case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
            evento = "<font color='purple'>BluetoothAdapter.SCAN_MODE_CONNECTABLE";
            log += logAdapter(uriData,evento);
            break;
        case BluetoothAdapter.SCAN_MODE_NONE:
            evento = "<font color='purple'>BluetoothAdapter.SCAN_MODE_NONE";
            log += logAdapter(uriData,evento);
            break;
        default:
            break;
    }
    logger = writeLog(log, "BluetoothAdapter.txt");
    evt.setEventLog(evento);
    dbHelper.saveEvent(evt);
}
if(action.equals(BluetoothAdapter.ACTION_CONNECTION_STATE_CHANGED)) {
    final int estado = intent.getIntExtra(BluetoothAdapter.EXTRA_CONNECTION_STATE, BluetoothAdapter.ERROR);
    evento = "<font color='red'>BluetoothAdapter.ACTION_CONNECTION_STATE_CHANGED :" + estado;
    log += logAdapter(uriData, evento);
    evt.setEventLog(evento);
    dbHelper.saveEvent(evt);
    logger = writeLog(log, "BluetoothAdapter.txt");
    showToast(showToastLog);
}
```

```
if(action.equals(BluetoothAdapter.ERROR)){
  log += String.valueOf(BluetoothAdapter.ERROR);
  evento = "<font color='red'>BluetoothAdapter.ERROR";
  log += logAdapter(uriData,evento);
  evt.setEventLog(evento);
  dbHelper.saveEvent(evt);
  showToastLog = "BluetoothAdapter.ERROR";
  logger = writeLog(log, "BluetoothAdapter.txt");
  showToast(showToastLog);
}
```

Fig. 5: Bluetooth Adapter States

The next state is to validate when finding other available devices within range, so we store the found device MAC and its NAME. Having the information about unconnected devices, available in the app.

```
if(action.equals(BluetoothDevice.ACTION_FOUND)){
  BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
  int rssi = intent.getShortExtra(BluetoothDevice.EXTRA_RSSI, Short.MIN_VALUE);
  Log.w("ACTION_ACL_CONNECTED","Entróa
ACTION_ACL_CONNECTED"+device.getName()+"\n"+device.getAddress()+"\n"+rssi);
  device dev = new device();
  dev.setId(0);
  dev.setName(device.getName());
  dev.setAddress(device.getAddress());
  dev.setContentDesc(String.valueOf(rssi));
  dev.setBloqueado(String.valueOf(false));
  dev.setTime(String.valueOf(Calendar.getInstance().getTime()));
  dev.setBonded(String.valueOf(false));
  dev.setHashCode(String.valueOf(device.hashCode()));
  boolean result = guardaDispositivo(dev);
  evento = "<font color='green'>BluetoothDevice.ACTION_FOUND";
  log += logAdapter(uriData,evento);
  evt.setEventLog(evento);
  dbHelper.saveEvent(evt);
  logger = writeLog(log, "BluetoothAdapter.txt");
  showToastLog = "BluetoothAdapter.ACTION_FOUND";
  showToast(showToastLog);
}
```

Fig. 6: Find available devices Function

Then, it validates the connection adapter status, so it validates the blocked devices, if the device is blocked, we can remove it from the device adapter's list, in other cases, we create the link between devices to pair them.

```
if(action.equals(BluetoothDevice.ACTION_ACL_CONNECTED)) {
  Log.d("ACTION_ACL_CONNECTED","Entró a ACTION_ACL_CONNECTED");
  log                +=                logAdapter(uriData,"BluetoothDevice."+BluetoothDevice.EXTRA_NAME+" -
"+BluetoothDevice.ACTION_ACL_CONNECTED);
  evento = "<font color='green'>BluetoothDevice.ACTION_ACL_CONNECTED";
  evt.setEventLog(evento);
  BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
  bloqueo = validaBloqueo(device.getAddress());
  UUID uuid = UUID.randomUUID();
  if(bloqueo){
    try {
      Log.w("ACTION_ACL_CONNECTED","Entró a removeBond");
      Method m = device.getClass()
          .getMethod("removeBond", (Class[]) null);
      m.invoke(device, (Object[]) null);
```

```
        evento = "<font color='red'>Dispositivo Bloqueado ("+uuid+"):
"+device.getAddress()+"; "+device.getName();
        log += logAdapter(uriData,evento);
        showToast("Dispositivo bloqueado ("+uuid+") : "+device.getAddress()+" : "+
device.getName());
      } catch (Exception e) {
        Log.e("ACTION_ACL_CONNECTED", e.getMessage());
        evento = "<font color='red'>Error de Dispositivo Bloqueado ("+uuid+"):
 "+device.getAddress()+"; "+device.getName();
      }
    }else{
      try {
        Log.d("ACTION_ACL_CONNECTED", "Start Pairing...");
        Method m = device.getClass()
            .getMethod("createBond", (Class[]) null);
        m.invoke(device, (Object[]) null);
        Log.w("ACTION_ACL_CONNECTED", "Pairing finished.");
        evento = "<font color='green'>Dispositivo Conectado ("+uuid+"):
"+device.getAddress()+"; "+device.getName();
        log += logAdapter(uriData,evento);
      } catch (Exception e) {
        Log.w("ACTION_ACL_CONNECTED", e.getMessage());
        evento = "<font color='red'>Error Dispositivo Conectado ("+uuid+"): "+device.getAddress()+";
"+device.getName();
      }
    }
      if(action.equals(BluetoothDevice.ACTION_ACL_DISCONNECTED)) {
        log += logAdapter(uriData,"BluetoothDevice."+BluetoothDevice.EXTRA_NAME+" –
"+BluetoothDevice.ACTION_ACL_DISCONNECTED);
        evento = "<font color='green'>BluetoothDevice.ACTION_ACL_DISCONNECTED";
        evt.setEventLog(evento);
        dbHelper.saveEvent(evt);
        logger = writeLog(log, "BluetoothAdapter.txt");
        showToastLog = "BluetoothDevice."+BluetoothDevice.EXTRA_NAME+" –
"+BluetoothDevice.ACTION_ACL_DISCONNECTED;
        showToast(showToastLog);
      }
    }
};
```

Fig. 7: Adapter State Connection Validation function

b) Device blocking

The state the device is found, is extremely important for the app. Therefore, if the app finds a device and this is active, when establishing a connection, it can be blocked to avoid negative actions to our device from this unwanted device. The following method blockDevice validates the state from this device and updates the blocked column from the Devices chart.

```
/*
 * Valida el estado del dispositivo y actualiza la columna bloqueado
 * */
public boolean blockDevice(device dev){
   boolean result = false;
   String estado = "bloqueado";
   try {
      String args [] = new String[1];
      args[0] = dev.getAddress();
      Cursor cursor =
this.getDevice(devicesContract.deviceEntry.tableName,null,devicesContract.deviceEntry.address+
"=?",args,null,null,null);
      if (cursor.getCount() > 0){
         cursor.moveToFirst();
         String bloqueado =
cursor.getString(cursor.getColumnIndex(devicesContract.deviceEntry.bloqueado));
         if(bloqueado == null){
            estado = "bloqueado";
         }
         else if(bloqueado.equals("bloqueado")) {
            estado = "desbloqueado";
         }
      }else{
         Log.e("deviceDBHelper", "Error bloqueando Dispositivo no
emparejado:"+dev.getAddress() );
      }
      ContentValues cv = new ContentValues();
      cv.put(devicesContract.deviceEntry.bloqueado, estado);
      SQLiteDatabase db = getWritableDatabase();
      db.update(devicesContract.deviceEntry.tableName, cv,
devicesContract.deviceEntry.address + "='" +
dev.getAddress()+"'", null);
      result = true;
   }catch (Exception e){
      Log.e("deviceDBHelper", "Error bloqueando Dispositivo: " + e.toString());
   }
   return result;
}
```

Fig. 8: Block Device Function

V. Testing Phase

a) Bluejacking attack phase

Attack deploy: To start we use the app BlueLock in the target device 2 with MAC A0:8D:16:88:A5:BD and block all the devices that have bluetooth activated. Then, we create a new contact in the attacker device 1 with MAC MAC C0:8C:71:84:94:A1 and it is sent or shared to the target device.

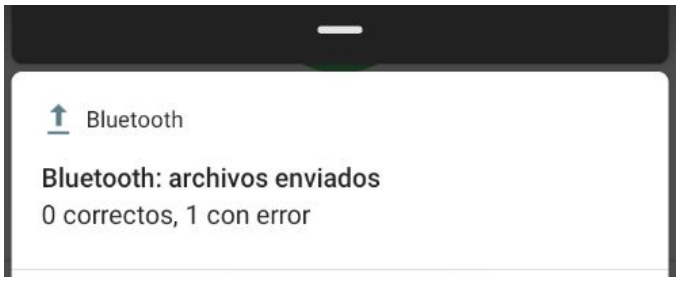

Fig. 9: Bluejacking Attack -Error message on contact transfer.

i. *Bluejacking attack results*

After executing the test, we can see the app blocks the contact sent, blocking all the communication channels between those devices.

b) Bluesnarfing attack test

i. *Deploying attack*

This attack is similar to the previous one, but it enters and steals information from the attacked device. We execute the same previous commands and we get the results shown in the console.

```
root@kali:~# sdptool browse --tree --l2cap C0:8C:71:84:94:A1
Failed to connect to SDP server on C0:8C:71:84:94:A1: Connection timed out
root@kali:~# sdptool browse --tree --l2cap C0:8C:71:84:94:A1
Failed to connect to SDP server on C0:8C:71:84:94:A1: Operation already in progress
root@kali:~# sdptool browse --tree --l2cap C0:8C:71:84:94:A1
Failed to connect to SDP server on C0:8C:71:84:94:A1: Connection timed out
root@kali:~#
```

ii. *Bluesnarfing attack results*

The attacked device avoids the description services reading and keeps inaccessible to the attacker device.

VI. Analysis Results

When we deployed the attacks and used the app Bluelock, we got a positive result due to the effectiveness when blocking attacker devices trying to deploy their attacks. In the bluejacking, it does not allow incoming notifications from the corrupted contact, and in the Bluesnarfing it was completely stopped since the device tried to start communication.

The app, through the blocking device functionality, offers better security and control of the device Bluetooth connections, because it allows the user to choose the devices that will be able to establish a connection and keep track of the bluetooth connections events. After testing the file transfer and Bluetooth attacks we see in the app the functionality filter for the bluetooth adapter states works as expected, so its functionalities could be expanded, defining protocols for every single state of the Bluetooth adapter, increasing the app's usability and security to protect our information from Bluetooth attacks.

The log registration allows us to access the Bluetooth adapter communication events in a short time range and to detect unusual events that take place in the Bluetooth interface range.

The functionality of the app consists in the device's bluetooth adapter control. In case the interface does not respond, it will need to be reseted to get control again, this action can be done by BlueLock.

An app that allows us to see the Bluetooth interface actions is not easily found, because they happen under a transparent communication cape for the user. So with the app BlueLock we can offer a better control of the use that bluetooth connections represent and the events at the moment of establishing connections.

V. Conclusions

The synergy between hacking and the bluetooth attacks, along with software development, allows them to complement each other, facilitating the design, development, testing and deploying of software solutions that improve security in device communication interface, and allow the implementation of closed mobile communication systems.

Most Bluetooth attacks are done while the device's interface is active, due to the protocol working as a receptor waiting for incoming communications and in that state attacks like Bluesnarfing can take place and steal information, taking advantage of the human factor to reach closeness of the target device.

Software solutions for mobile devices focused on connection validation and data packages can improve Android devices' security in places like

apartment buildings where there are a lot of mobile devices active and we have a higher chance to be hacked in such environments. BlueLock is a security app oriented to connectivity events that allows unwanted device detection and offers the chance to block them at any time. This is possible by validating the Bluetooth interface state through Intent filters, that is sensible to the Bluetooth adapter changes.

The robustness of the testing set for BlueLock solution, has given positive security results that allow us to block BLuejacking or Bluesnarfing attacks coming from previously blocked devices. In conclusion, we are allowing the user to be responsible for managing the device he allows or not to connect.

References Références Referencias

1. Ramos, A., Barbero, C., Martínez, R., García, A. y González, J. (2015). *Hacking y seguridad de páginas web*. Colombia: Ediciones U, Ra – Ma.
2. Nolasco, J. (2016). Desarrollo de aplicaciones móviles con Android. Segunda edición. Colombia: Ediciones U, Ra – Ma.
3. Ciampa, M. (2018). *Security+ Guide to Networks Security Fundamentals*. Sexta edición. Boston, USA: Cengage.
4. Haataja. K., Hypponen, K. Pasanen, S. y Toivanen, P. (2013). *Bluetooth Security Attacks: Comparative analysis, attacks and countermeasures*. Finlandia: Springer.
5. Conkin, A., White G., Cothren, C., Davis, R., y Williams, D., (2018). *Principles of Computer Security: CompTIA Security+ and Beyond*. USA: McGraw Hill Professional.
6. Minar, N. y Tarique, M. (2012). Bluetooth Security Threats and Solutions: A Survey. International Journal of Distributed and Parallel Systems (IJDPS). 3(1). Recuperado de https://www.researchgate.net/publication/267200901_Bluetooth_Security_Threats_And_Solutions_A_Survey
7. Weed, M. (2018). Bluetooth IoT Applications: From BLE to Mesh. USA: IoT for all. Recuperado de https://www.iotforall.com/bluetooth-iot-applications/
8. García, C. (2010). Hablemos de Spoofing. Hacking Ético. Recuperado de https://hacking-etico.com/2010/08/26/hablemos-de-spoofing/
9. OCCUPYTHEWEB. (2016). The Hacks of Mr. Robot: How to Hack Bluetooth. Los Angeles, California: Wonder How To. Recuperado de https://null-byte.wonderhowto.com/how-to/hacks-mr-robot-hack-bluetooth-0163586/
10. Mitra, A. (2017). What is Bluestack Attack?. The Security Buddy. Recuperado de https://www.thesecuritybuddy.com/bluetooth-security/what-is-blue smack-attack/2/
11. Android. (2019). Conoce android studio. USA: Developers Android. Recuperado de https://developer.android.com/studio/intro/?hl=es-419
12. Android. (2019). Documentation, android.bluetooth. USA: Developers Android. Recuperado de https://developer.android.com/reference/android/bluetooth/package-summary.
13. Alma del Cid, Méndez R. y Sandoval F. (2011). *Investigación. Fundamentos y metodología*. Segunda edición. México: Pearson.
14. ADN Sureste. (2019). Este "error" en el Bluetooth pone en riesgo tu celular. México. Recuperado de https://www.adnsureste.info/este-es-error-en-el-bluetooth-pone-en-riesgo-tu-celular-2130-h/.
15. Clarín. (2019). Advierten sobre los peligros de utilizar el Bluetooth de tu celular. Buenos Aires, Argentina. Recuperado de https://www.clarin.com/tecnologia/advierten-peligros-utilizar-bluetooth-celular_0__WpofinjL.html.
16. Álvarez, R. (2017). Se descubre una "grave vulnerabilidad" en Bluetooth que deja expuestos los dispositivos a posibles ataques. España. Recuperado de https://www.xataka.com/seguridad/se-descubre-grave-vulnerabilidad-bluetooth-que-deja-expuestos-dispositivos-a-posibles-ataques.
17. Carro, G. (2019). Esto es lo que puede pasar si tienes el bluetooth del móvil siempre conectado. Revista GQ. España. Recuperado de https://www.revistagq.com/noticias/articulo/bluetooth-movil-siempre-conectado-peligros-hackers.