A defensive man-in-middle approach to filter BLE packets

Ahmed Aboukora, Guillaume Bonnet, Florent Galtier, Romain Cayre, Vincent Nicomette, Guillaume Auriol

DEMO: A defensive man-in-middle approach to filter BLE packets

Ahmed Aboukora
Guillaume Bonnet
Université de Toulouse, IRIT/INP, ENSEEIHT, Toulouse, France
firstname.lastname@etu.enseeiht.fr

Florent Galtier
Romain Cayre
CNRS, LAAS, Toulouse, France
firstname.lastname@laas.fr

Vincent Nicomette
Guillaume Auriol
Univ de Toulouse, INSA, LAAS
Toulouse, France
firstname.lastname@laas.fr

ABSTRACT

In this paper, we propose an original defensive strategy in which we benefit from the use of Man-in-The-Middle attacks in order to protect some vulnerable BLE devices. More precisely, we describe a tool that uses a Man-in-The-Middle attack to implement a wireless firewall for BLE communications, that is able to block specific commands, make some services invisible on BLE devices, or to force out weak pairing mechanisms.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**.

ACM Reference Format:
Ahmed Aboukora, Guillaume Bonnet, Florent Galtier, Romain Cayre, Vincent Nicomette, and Guillaume Auriol. 2021. DEMO: A defensive man-in-middle approach to filter BLE packets. In *14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), June 28–July 2, 2021, Abu Dhabi, United Arab Emirates.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3448300.3468259

1 MOTIVATIONS

Wireless communication protocols are becoming more and more widespread in everyday devices. Among those protocols, the Bluetooth Low Energy, or BLE, a lightweight and power-efficient variation of Bluetooth, is spreading particularly fast, as it is becoming omnipresent in mobile phones or on computer wireless boards.

However, even if BLE specification defines diverse security mechanisms, they are not activated or used in a vast majority of devices using it[6]. Hence, many BLE devices are vulnerable to various attacks, in particular Man-in-The-Middle attacks[2, 4].

In this paper, we explore the feasibility of turning an offensive Man-in-the-Middle strategy into a defensive monitoring system, allowing to filter BLE traffic. The filtering of packets generally requires a central node in which the communications can be monitored and analysed, in order to be dropped or forwarded, which is quite difficult to obtain in BLE due to several constraints: 1) BLE communications are peer to peer, and thus don't involve such a central node by design, and 2) the Bluetooth Low Energy stack includes a channel hopping algorithm, complicating passive monitoring approaches based on SDR or sniffers.

As a consequence, a Man-in-the-Middle defence strategy allows to solve these problems by introducing a reliable way to monitor BLE traffic by means of an intermediate node in the communication, which can apply different filtering techniques. Being able to analyse and filter BLE packets allows to mitigate various attacks and improve the security level of this protocol. This can be done by dropping suspect applicative packets, selectively removing sensitive services or characteristics from a Slave's ATT server, and controlling which security mechanisms are allowed, to ensure that no flawed methods are in use. Some detailed information on this filtering approach are provided in the next Section.

2 MAN-IN-THE-MIDDLE-BASED FIREWALL

2.1 BLE Man-in-The-Middle

The Bluetooth Low Energy connections work in a Master-Slave model, where a Master connects to a Slave, which then acts as a server exposing different services via the Attribute Protocol (ATT) and Generic Attribute Profile (GATT), defining services and characteristics for the device.

Let us consider a legitimate BLE Master M connecting to a legitimate BLE Slave S. A Man-in-the-Middle attack consists in introducing a BLE object F, that connects to the Slave pretending to be the Master, then exposes to M a fake Slave with the same address as the legitimate one. When M connects to F, the latter can either block, modify or simply retransmit the frames to S. Indeed, S is following its connection with F, while M initiates an independent connection with F, following another channel sequence. More detailed descriptions of Man-in-The-Middle attacks in BLE can be found in the publications of S. Jasek [4] or D. Cauquil [2].

2.2 Firewall description

The principle of our firewall approach is based on the capabilities of the attacker in the context of such a Man-in-The-Middle attack. The firewall device acts as a transparent proxy between the Master and Slave devices, controlling entirely all communications of the connection. The main capabilities of our firewall are described hereafter.

Blocking specific packets: BLE communications use several types of packets, in particular commands linked to the ATT/GATT layers, referring to a given service or characteristic of the Slave by using an identifier called *handle*. As most of the existing connected objects don't activate the BLE security mechanisms[6], an attacker can easily reverse-engineer the applicative protocol and identify actions or sensor data related to *handles*. We then implemented in the firewall filter measures to allow or deny specific types of packets, of packets with specific *handles* or *values*, to filter out unusual packets that may be used to perform an attack.

Hide services or characteristics: some services on the devices may expose sensitive data, or their modification may rise security issues. The firewall is able to hide those services from the Master device, by removing them from the answers to discovery procedures, and exposing a reduced version of the original server to the Master.

Forbid weak pairing mechanisms: the BLE specification describes different pairing methods, depending on the protocol to exchange a temporary key used to initiate the establishment of secure communications. The security of the following communications may then greatly depend on the method that was used for this exchange. However, some of these mechanisms are flawed[5], or can be manipulated to decrease security[1]. We then added to the firewall the possibility to intercept the pairing procedure to block weak pairings.

3 FILTERING RULES

Our BLE firewall can be configured by means of two categories of filtering rules. The first category is aimed at filtering BLE packets according to their type and their fields. They are relevant to block some specific BLE commands (such as commands corresponding to write or read access to resources or weak pairing as explained above). Such rules are configured between a specific header and footer (`BLE_TABLES` and `END_BLE_TABLES`). The second category aims at hiding some services or characteristics of BLE Slave objects at the GATT abstraction level. Such rules are configured between a specific header and footer (`GATT_FILTER` and `END GATT_FILTER`). This Section provides an overview of the rules syntax for each category, as well as simple examples and ends with some implementation details.

3.1 Filtering rules for BLE packets

The rules of the first category as structured as follows:

```
action <allow/deny> type <packet_type> [<name> <value>, ...]
```

where :

- `action <allow/deny>` represents what the firewall is supposed to do when a packet matches the following rule: `allow` to forward the packet or `deny` to drop the packet
- `type <packet_type>` identifies the type of BLE packet concerned by the rule;
- `<name> <value>` identifies any field of the packet, through the name of the field as well as its value.

The list of (`<name> <value>`) couples depends on the packet type. The purpose of this paper is not to provide an exhaustive list of these fields, but rather some examples. For instance, the rule:

```
action deny type BLEWriteCommand handle 0x29 value 0x0
```

is intended to drop the packets of type `BLEWriteCommand` with the `handle` 0x29 and the `value` 0x0. The idea behind this rule is to prevent any attacker from modifying this specific characteristic of the object with the value `0x0`.

In the same way, this category can also be used to intercept some weak pairing mechanisms. For example, the following rules:

```
action deny type BLEPairingRequest mode LELegacyPairing
action deny type BLEPairingRequest mode LESecurePairing
  method JustWorks
action deny type BLEPairingRequest mode LESecurePairing
  minkeySize 16
```

prevent any use of legacy pairing, JustWorks pairing and pairing methods using a cryptographic key of a size inferior to 16.

Our filtering mechanism includes a default action (forward or drop) for all packets that do not match any filtering rules configured: `default <allow/deny>`.

3.2 Filtering rules for GATT servers

The rules of the second category are intended to protect a BLE slave that exposes some services through a GATT interface. More precisely, they are designed to filter services and characteristics to hide them from the Master's point of view.

These rules are structured as follows:

```
entity GATT type <GATT_type> [<name><value> ...]
```

where:

- `type <GATT_type>` represents the type of information that is hidden. Possible values are `Service`, `Characteristic` and `Descriptor`
- `<name> <value>` identifies some fields characterising the hidden resource

The list of (`<name> <value>`) couples depends on the resource considered. Once again, our purpose is not to provide an exhaustive list of these fields in this paper. They mainly consists of *handles*, *uuid*, *values* and *permissions* which correspond to the official terminology at the GATT level to describe some resources.

Let us take the example of a MacBookAir laptop. As presented in Figure 1, the laptop acts as a BLE Slave that exposes different GATT *services* and the service 'Device Information' exposes two *Characteristics* corresponding to the 'Manufacturer Name String' and the 'Model Number String'.

The first following GATT filtering rule is used to 'hide' the whole *service* 'Device Information', while the second only 'hides' the second *characteristic* from the Master point of view.

```
entity GATT type Service serviceType primary
uuid 0x180a startHandle 0x0010 endHandle 0X0014
entity GATT type Characteristic declarationHandle 0x0011
uuid 0x2a29 valueHandle 0x0012
```

3.3 Implementation details

The BLE firewall was implemented using Mirage[3], which is a powerful and modular framework dedicated to the security analysis of wireless communications. More precisely, our implementation benefits from the use of the `ble_mitm` module, to which we added filtering capabilities by using Mirage's *scenarios*, triggering custom callbacks on receiving specific packet types. Those *scenarios* are generated by parsing configuration files containing the above rules, and automatically generating the corresponding callbacks that implement the expected filtering behaviour.

4 USE CASE SCENARIOS

This Section describes two use case scenarios that we implemented to show the relevance of our filtering strategy.

```
┌Services──────────────────────────────────────────────────────────────────────────────────┐
│ Start Handle │ End Handle │ UUID16 │ UUID128                          │ Name               │
├──────────────┼────────────┼────────┼──────────────────────────────────┼────────────────────┤
│ 0x0001       │ 0x0005     │ 0x1800 │ 0000180000001000800000805f9b34fb │ Generic Access     │
│ 0x0006       │ 0x0009     │ 0x1801 │ 0000180100001000800000805f9b34fb │ Generic Attribute  │
│ 0x0010       │ 0x0014     │ 0x180a │ 0000180a00001000800000805f9b34fb │ Device Information │
│ 0x0020       │ 0x0023     │        │ d0611e78bbb44591a5f8487910ae4366 │                    │
│ 0x0024       │ 0x0027     │        │ 9fa480e0496745429390d343dc5d04ae │                    │
└──────────────┴────────────┴────────┴──────────────────────────────────┴────────────────────┘

┌Service 'Device Information'(start Handle = 0x0010 / end Handle = 0x0014)──────────────────────────────────────────────────────────────┐
│ Declaration Handle │ Value Handle │ UUID16 │ UUID128                          │ Name                     │ Permissions │ Value          │
├────────────────────┼──────────────┼────────┼──────────────────────────────────┼──────────────────────────┼─────────────┼────────────────┤
│ 0x0011             │ 0x0012       │ 0x2a29 │ 00002a2900001000800000805f9b34fb │ Manufacturer Name String │ Read        │ Apple Inc      │
│ 0x0013             │ 0x0014       │ 0x2a24 │ 00002a2400001000800000805f9b34fb │ Model Number String      │ Read        │ MacBookAir7,2  │
└────────────────────┴──────────────┴────────┴──────────────────────────────────┴──────────────────────────┴─────────────┴────────────────┘
```

Figure 1: GATT services of the MacBookAir and characteristics of 'Device Information' service

4.1 Smart keyring use case

A BLE smart keyring (`Itag` model in this experiment) is able to emit a sound when contacted by a smartphone and thus can be easily located when lost. Usually, the smartphone sends a BLE packet to the keyring to locate it but the keyring can do the same with the smartphone. In this use case, we configured the filtering rules that follow in order to: 1) prevent the keyring from ringing the smartphone (by denying in line 5 a specific `BLEHandleValueNotification` packet) but allow the smartphone to do so (by authorising in lines 3 and 4 specific `BLEWriteCommand` packets) and dropping all other command packets (line 6) ; 2) make it impossible to invoke the service 'Device Information" (line 9).

```
1 BLE_TABLES
2 TARGET FC:58:FA:xx:yy:zz
3 action allow type BLEWriteCommand handle 0x29 value 0x2
4 action allow type BLEWriteCommand handle 0x29 value 0x0
5 action deny type BLEHandleValueNotification handle 0x25 value 0x1
6 default deny
7 END BLE_TABLES
8 GATT_FILTER
9 entity GATT type Service serviceType primary uuid 1800 endHandle 0x0005
10 END GATT_FILTER
```

A short demonstration video can be found at https://homepages.laas.fr/nicomett/Videos/defensive_BLE_MiTM_demo.mkv. The functionality which is filtered here is not critical from the security point of view but it was intended to illustrate what can be done with such filtering rules. They could of course be used to filter some critical functionalities, such as the update over-the-air.

4.2 Smartwatch use case

This use case corresponds to the prevention of a real attack targeting a smartwatch. This smartwatch offers several services, and can be used as a personal assistant, a fitness tracker or health monitor. It can also receives SMS from the smartphone. As this smartwatch does not use, by default, any encryption mechanisms, it is possible for an attacker to inject specific BLE packets allowing to inject fake SMS messages. The following filtering rules aims at preventing such attacks:

```
action deny type BLEWriteRequest handle 0xb value '6f7171'
action deny type BLEWriteRequest handle 0xb value '8f'
```

They consist in dropping BLE commands with a specific payload matching the '6f7171' pattern (used as headers on injected packets) or the '8f' pattern (used as footers on injected packets) which characterize the SMS injection attack.

5 DISCUSSION

In this paper, we described a prototype BLE firewall, based on the defensive use of a Man-in-The-Middle approach to filter out specific frames or hide elements exposed by connected devices, and some experiments that we successfully carried out with this prototype. As this prototype is a preliminary research work, some open questions still remain and we are currently working on some enhancements. First, BLE handles need to be a continuous sequence, making it necessary to change the handles after the ones we filtered out for discovery procedures to function normally. However, some applications don't use dynamic discovery to get the handles, and instead use hard-coded handles to access characteristics and services. While this could be circumvented by detecting which application works using dynamic discovery, and which does not, an attacker could exploit this, for example to know which handles were removed from the server. Second, our approach is based on the principle that our firewall is the first to connect to the Slave, because a majority of BLE Slave devices only allow for one connection at a time. This makes it possible for an attacker to connect first to some devices and prevent the firewall from intercepting the BLE communications and thus detecting any potential attack. Finally, such a firewall strategy can be deactivated by an attacker jamming the wireless communications between the firewall and the legitimate Slave, resulting in a disconnection between the devices, and allowing the attacker to connect before the firewall restarts its connection, hence bypassing it. As future work, we plan to work on anti-jamming mechanisms to complement our approach and consider the risks associated with this specific attacker model.

REFERENCES

[1] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B Rasmussen. 2019. The {KNOB} is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1047–1061.

[2] Damien Cauquil. 2016. Btlejuice: The bluetooth smart mitm framework. *DEF CON* 24 (2016), 4–7.

[3] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Eric Alata, Mohamed Kaaniche, and Geraldine Marconato. 2019. Mirage: towards a Metasploit-like framework for IoT. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 261–270.

[4] Sławomir Jasek. 2016. Gattacking Bluetooth smart devices. In *Black hat USA conference*.

[5] Mike Ryan. 2013. How Smart is Bluetooth Smart. *SchmooCon 9* (2013).

[6] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. 2019. Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1469–1483.