Who's Accessing My Data? Application-Level Access Control for Bluetooth Low Energy

Pallavi Sivakumaran(✉) and Jorge Blasco

Royal Holloway, University of London, Egham, UK
{pallavi.sivakumaran.2012,jorge.blascoalis}@rhul.ac.uk

Abstract. Bluetooth Low Energy (BLE) is a popular wireless technology deployed in billions of devices within the Internet-of-Things (IoT). The data on these devices is often related to user health or used to control safety-critical functionality, which makes it vital to protect the data from unauthorised access or manipulations. The only mechanism that is fully defined within the BLE specification for protecting sensitive data is *pairing*. This occurs at the device-level rather than at the application-level, and leaves BLE data vulnerable to unauthorised access at higher layers. When a BLE device interacts with a multi-application platform (i.e., a device that hosts more than one application, such as a mobile phone), when one application is able to access data from the BLE peer, all other applications on the same multi-application platform are also implicitly allowed the same access. The solutions suggested thus far for this vulnerability are either impractical for most users, not backward compatible with billions of existing devices, or do not suit normal BLE usage scenarios. In this paper, we conduct an analysis considering practical aspects regarding the BLE ecosystem, and thereafter propose a solution that will extend the available protection for BLE data to the application layer. Our solution ensures protection by default for BLE data, and is entirely backward compatible with existing BLE implementations, requiring no modification to resource-constrained BLE peripherals or companion applications. We also present an open-source proof-of-concept implemented on the Android-x86 platform. This, when tested against experimental and real-world devices and applications, demonstrates the viability and efficacy of our proposed solution.

Keywords: Bluetooth low energy · Application-level security · Multi-application platforms · GATT

This research has been partially sponsored by the Engineering and Physical Sciences Research Council (EPSRC) and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 399, pp. 252–272, 2021.
https://doi.org/10.1007/978-3-030-90022-9_13

1 Introduction

The Bluetooth Low Energy (BLE) technology enables compatible devices to exchange small amounts of discrete data over a wireless interface, in an energy-efficient manner. Typical usage scenarios, particularly in consumer applications, involve communications between a resource-constrained "peripheral", such as a fitness tracker or glucose monitor, with a more powerful "platform device", such as a mobile phone or personal computer (PC). More precisely, the communication will be between the BLE peripheral and an *application on* the phone or PC.

The Bluetooth specification, defined by the Bluetooth Special Interest Group (SIG), provides optional mechanisms for restricting access to BLE data on a per-device basis, via Bluetooth pairing. However, despite BLE being a full-stack protocol [28], the specification does not provide concrete, well-defined mechanisms for restricting access to BLE data on a per-*application* basis. This lack of application-level restrictions is problematic on devices such as mobile phones and computers, which normally host multiple applications. On such devices, when one application is able to access data from a BLE peripheral, all other applications are implicitly given access to the same data, even if the BLE data is protected by pairing [25]. This vulnerability makes it possible for unauthorised applications to access sensitive data from BLE peripherals. Depending on the BLE device, this could have serious consequences for user privacy (e.g., if user health data is leaked from fitness or medical devices) or safety (e.g., if a BLE door lock or eScooter is manipulated).

Previous suggested solutions include implementation of application layer security by developers or modification of the mobile platform [1,18]. However, these do not enable protection by default across all possible implementations of the BLE technology. Platforms such as Android and iOS currently implement restrictions on applications by requiring the granting of some permissions by the user [1,4]. However, these restrictions are not present on other multi-app platforms and, in any case, do not restrict access on a per-peripheral basis.

A solution that is proposed to mitigate this vulnerability must take into consideration not only technical limitations but also practical issues. That is, a highly secure solution is not particularly useful if it will not be implemented in the vast majority of BLE deployments. In this work, we define security and system requirements based on typical BLE configurations, usage mechanisms and user involvement. We also conduct a pragmatic multi-faceted stakeholder analysis. Based on these factors, we propose a specification-level solution, which meets the requirements that we have set out. Our solution achieves protection by default, minimal overhead for resource-constrained devices, and full backward compatibility with existing BLE systems.

To illustrate the viability and efficacy of our solution, we implement a Proof-of-Concept (POC) on the Android-x86 platform. Our POC demonstrates that, with practicable modifications to the multi-application platform, covert data access attempts are brought to the attention of users and are therefore defeated.

Key Contributions. In summary, our contributions are as follows:

- We set out requirements (Sect. 3) for a solution to the unauthorised data access vulnerability that is present in multi-application BLE platforms (Sect. 2).
- We perform a stakeholder analysis with practical considerations, and determine that an asymmetric specification-level modification ensures greatest coverage for a solution (Sect. 4).
- We describe (Sect. 5), evaluate (Sect. 6), and implement an open-source POC for (Sect. 8) a solution that involves minimal changes to the BLE stack.
- We detail benefits realised by our solution in addition to those implied by our set requirements (Sect. 7).

2 Background: Unauthorised Data Access in BLE

BLE is a full-stack protocol, which comprises three main subsystems: the Controller, the Host and Applications [28]. The functionality of the Controller and Host are defined within the core Bluetooth specification, while "applications" are defined in terms of *profiles* within individual specifications (e.g., Heart Rate Profile or Glucose Profile specification). Profile specifications in turn reference one or more *service* specifications. The core specification as well as profile and service specifications are defined and maintained by the Bluetooth SIG.

Each service referenced by a profile contains a number of *characteristics*, where a characteristic holds a single discrete piece of information. Services and characteristics are both types of BLE *attributes*, governed by the Attribute Protocol (ATT). The hierarchical structuring of attributes into characteristics and services is controlled by the Generic Attribute Profile (GATT).

Attributes can have different permissions applied to them: *access permissions* dictate whether an attribute can be read/written; *authentication/encryption permissions* indicate whether an authenticated/encrypted link (achieved via pairing) is required between the two devices before the attribute can be accessed; *authorisation permissions* indicate whether client authorisation is required.

SIG-defined service/profile specifications define security considerations in terms of the different permissions. For example, the Glucose Profile states "*All supported characteristics specified by the Glucose Service shall be set to Security Mode 1 and either Security Level 2 or 3*". The "*Security Mode 1... Level 2 or 3*" refers to encryption and authentication permissions. To satisfy the security requirements defined in the Glucose Profile, implementing devices (for example, a glucose meter and a mobile phone) must undergo pairing.[1]

Pairing takes place between the lower layers of the two devices, and does not extend to the application layer. If the mobile phone in this example hosted a single, trusted application, then the security requirements specified in the Glucose Profile might be sufficient to protect the glucose measurement data from unauthorised access.[2] However, mobile phones typically host numerous,

[1] The communicating devices will normally also *bond*, i.e., store long term keys.

[2] For the purpose of our discussion, we assume a secure pairing process.

potentially untrusted applications. Therefore, the fact that the Glucose Profile does not require restricting access at the application level means that, once one application on a mobile phone has triggered pairing (and bonding) with the glucose meter, any other application on the mobile phone will also be able to access the glucose measurement data. We have previously demonstrated that such access is possible in a covert manner on the Android platform [25].

We observe that authorisation permissions *can* restrict access at the application layer. However, implementing these permissions can reduce flexibility and interoperability by tying a BLE device to a single application (where the user may desire a choice). Also, these permissions are not required in any SIG-defined service specification except the Insulin Delivery Service. Further, the mechanism of *how* to accomplish client authorisation is left up to the developer. A large number of existing BLE devices either do *not* implement authorisation permissions or implement them in an insecure manner [25].

Note that the problem of unauthorised data access is present only when a platform hosts multiple applications, as with mobile devices. However, the vulnerable *data* tends to be that obtained from BLE peripherals, which typically host only a single application.

3 Environment

In this section, we outline our threat model (Sect. 3.1) and define a set of security and system requirements (Sects. 3.2, 3.3) that we believe should apply to any solution that addresses the problem of unauthorised BLE data access on multi-app platforms.

3.1 Threat Model

We make the assumption that any app on the multi-application platform issues BLE requests via a BLE stack implemented by the platform, i.e., the application cannot circumvent the stack that is implemented by the platform. We also assume that the application cannot directly access the components of the platform-implemented BLE stack or influence its operations by any means other than via robust platform APIs. In addition, we assume an honest and uncompromised platform and peripherals. However, *applications* are from multiple third-party developers and are assumed to be untrusted. These applications may abuse the unauthorised data access vulnerability to obtain and manipulate data being stored on a BLE peripheral without the user's knowledge and consent. This is the kind of behaviour our solution aims to protect against.

3.2 Security Requirements

To protect against unauthorised data access attacks at the application-layer, we define three main security requirements. These are based on typical multi-application platform configurations and usage, and the shortcomings that we have identified with existing restriction mechanisms.

SecRQ1: Prevention of Unauthorised Access to BLE Data. An application should not be able to access the data from a BLE peer device without the user's knowledge and explicit authorisation. Note that the term "authorised" in this context should not be confused with the "authorisation permissions" already defined within the Bluetooth specification.

SecRQ2: Per-Device Access Control. User authorisation should be granted to an application for every peer device *individually*. That is, if an application is granted permission to access one BLE peer device, it should not automatically be possible for that application to access any other BLE peer device.

SecRQ3: Access Revocation. The user must be able to revoke access that has previously been granted to an application for a BLE peer device. This limits the exposure of data in the event of late identification of malicious app behaviour.

3.3 System Requirements

There are billions of BLE-enabled devices in use today [6] and most are in consumer applications, where the end users are not necessarily highly technical. This results in a need for security solutions that do not require high levels of user involvement. In addition, most BLE peripherals are resource-constrained by design and will not be able to handle large amounts of processing. This makes complex cryptographic protocols less desirable. While these factors are not directly related to the security of a system, they need to be taken into consideration when proposing a security solution. We therefore define three key *system* requirements, bearing in mind user involvement, the number of BLE devices extant in the world today and the asymmetric nature of resources on communicating BLE devices.

SysRQ1: Protection by Default. All devices that implement (the modified version of) BLE should incorporate protection by default. Any specification-compliant BLE system should automatically protect against and be protected from unauthorised data access at the application layer, without the need for additional user intervention (beyond the explicit granting of permissions or authorisation).

SysRQ2: Backward Compatibility. Devices that incorporate the solution should function with existing devices. Given that billions of BLE-enabled devices exist today, a solution that obsoletes such a vast number of devices would not be acceptable.

SysRQ3: Minimal Overhead for Resource-Constrained Devices. The solution should not incur a significant processing overhead for the more resource-constrained device, as this would lead to greater power requirements and quicker battery drain, thereby defeating the purpose of BLE.

4 Devising a Solution Strategy

The requirements we have described in Sects. 3.2 and 3.3 are necessary for a secure and utilitarian solution to the unauthorised data access problem. However, the most secure solution is of no value if it will not be applied to a large proportion of the BLE ecosystem due to lack of technical capability or stakeholder involvement. In this section, we discuss the primary stakeholders in the BLE ecosystem and describe practical considerations that should be taken into account when proposing a solution. From this, we determine the most suitable solution strategy to ensure maximum coverage.

4.1 Stakeholders Within BLE

There are five primary stakeholders within the BLE ecosystem:

1. The *Bluetooth SIG* defines and maintains the Bluetooth specification, as well as BLE services and profiles (such as the Glucose Profile mentioned in Sect. 2).
2. *Chipset vendors* produce BLE-enabled chipsets, which are then used in platforms and peripherals. Chipset developers may also provide BLE stacks for their products, to enable developers to create BLE end products quickly and easily. Examples include Qualcomm, NXP, Nordic Semiconductor, Texas Instruments, STMicroelectronics, etc.
3. *Platform vendors* develop and maintain BLE-enabled platforms, typically supporting multiple applications. Prominent examples are Android, iOS/Mac OS, Windows and Linux.
4. *Developers* manufacture BLE-enabled end products (e.g., fitness trackers, medical monitoring devices, eScooters, smart locks). They normally also develop companion apps (that typically run on multi-app platforms) to interface with their products.
5. *Users* are the ultimate consumers for BLE-enabled products and services.

Users are only considered in terms of the impact of the vulnerability and the ease of applying a solution. Users are at most expected to update their devices' operating system or firmware and provide explicit authorisation to applications. We do not expect users to *implement* any part of a solution. We therefore confine the discussion on implementation to the first four entities.

4.2 Practical Considerations

When a BLE security solution is proposed, the likelihood of it being implemented depends on a number of factors. In this section, we analyse those factors in terms of the involved stakeholders.

Number of Entities. The likelihood of a solution being implemented depends in part on the number of entities that are required to implement it. The smaller the number, the easier it is to communicate the solution to them and the greater the reach of the solution. When considering BLE stakeholders in terms of numbers, the SIG is a single entity (albeit made up of a large number of members). This makes it a single point of communication, from which the solution

will trickle down to implementing entities (platform vendors, chipset vendors and developers). There are a limited number of platform vendors, and the four most prominent platforms (Android, Windows, iOS, and Mac OS) account for over 95% of the worldwide OS market share [27]. BLE chipset vendors are more numerous than platform vendors, but not by a large margin (15–20 vendors [16]). Developers, on the other hand, are multitudinous (several hundreds [7]); it would therefore be very difficult to communicate a solution to all possible developers.

Stakeholder Participation. Not all stakeholders respond satisfactorily when they are made aware of a vulnerability. The security behaviour of a stakeholder tends to be associated with the prominence of the stakeholder (in terms of brand value, which may act as an incentive to adopt strong security practices), as well as the availability of organisational support, knowledge and resources [5,11,15]. The SIG and most platform/chipset vendors have clear, mature processes in place for vulnerability reporting, assessment and mitigation, whereas many developers may not even respond when informed of issues [20]. Further, in interdependent ecosystems such as BLE, where platforms and chipsets implement the specification, and developers create end-products on top of the platforms and chipsets, there is a certain degree of "responsibility relaying". That is, each stakeholder presumes that the responsibility for implementing the solution belongs to another stakeholder. This phenomenon of "passing the buck" is prevalent within IT security, with responsibility being transferred down the supply chain or to other stakeholders [21,24]. In the case of BLE, we postulate that only a specification-level change will induce most of the remaining stakeholders within the BLE ecosystem to incorporate a solution; the solution would have to be implemented in order to claim conformance with the specification.

Availability of Update Mechanism. Stakeholder participation, as described in preceding sections, is key to solution implementation, but equally so is the availability of an actual mechanism for performing the implementation. In the case of the Bluetooth specification, all updates are to a document, which can be updated in a straightforward manner. The solution must thereafter be implemented by the remaining stakeholders. Platform devices, such as mobile phones and computers, have fairly robust update mechanisms. Therefore, a solution implementation can be easily rolled out on these devices. Most modern BLE chipsets support over-the-air (OTA) firmware updates, enabling updates to applications and sometimes also to the BLE stack. However, many IoT devices do not incorporate such update mechanisms [30], which means that a large proportion of existing BLE *peripherals* cannot be modified.

4.3 Discussion

Based on the large number of BLE end-product developers, the lower likelihood of developer participation, and the lack of firmware update mechanisms in many BLE peripherals, we reach the conclusion that a security solution that does

not require involvement from end-product developers is more likely to actually be implemented. We also observe that, because a single platform device normally communicates with multiple peripherals, an asymmetric solution involving changes to only the platforms (which are far fewer in number) will be an effort-efficient way to mitigate the unauthorised data access vulnerability for a larger proportion of the BLE ecosystem. Further, according to Sect. 4.2, a specification-level change is more likely to prompt changes to platform devices than individual communications with platform vendors. In addition, a specification-level change ensures security by default even if new BLE-enabled multi-app platforms are introduced in the future (without the need for communicating the solution to each new platform vendor individually).

5 Proposed Solution

In accordance with our analysis in Sect. 4, our proposed solution involves changes to the BLE stack and primarily involves modifications to multi-application platforms. Our solution also requires minor changes to the peripheral which, however, can be avoided while still retaining the expected outcome (see Sect. 7).

Our solution introduces three new BLE components/properties:

1. **ATT Access Database (AAD):** A database for storing application access permissions.
2. **ATT Access Manager (AAM):** A layer within the BLE stack, responsible for performing the main access control functions.
3. **Device/Platform Mode:** A property for a BLE system, which controls the behaviour of a BLE device with respect to the new functionality.

Sections 5.1 to 5.3 describe the purpose and, if relevant, the functionality of each of these elements in detail. Section 5.4 discusses concerns relating to user authorisation, while Sect. 5.5 describes access revocation.

5.1 The ATT Access Database (AAD)

The AAD stores per-device (i.e., BLE peer), per-application *access records*, as authorised by the user, for all applications that have made a GATT request for a BLE peer device, and for all such BLE peer devices connected to the platform. An access record has three components:

1. AppID: A unique identifier for the application that has made a GATT request to the platform. This must be assigned by the platform. It should not be possible for the application to manipulate its AppID.
2. DeviceID: A unique identifier for the BLE peer, e.g., the hardware address.
3. Permission: A value indicating whether access for an app (identified by AppID) to a BLE device (identified by DeviceID) is *Allowed* or *DenyListed*.

By default, records do not exist for an application until the application makes a GATT request. The first time a record is added to the AAD for an application-device pair, the associated permission will be as selected by the user. This is described in detail in Sect. 5.2.

Positioning of the AAD. Similar to the Security Database used within the existing design of BLE, the AAD does not feature within our modified BLE *stack*, but must be implemented by the platform in order for the BLE system to be operational. The AAD only communicates with the AAM. Therefore, the functionality of the AAD can also be subsumed into the AAM.

Access by Applications. The AAD must not be accessible to higher layer applications. It should not be possible for an application to query its AAD permissions or to add itself to the AAD, as that would defeat the access control mechanisms that are in place.

5.2 The ATT Access Manager (AAM)

We introduce the AAM as a new layer within the BLE Host sub-system. It serves as an access control mechanism for GATT requests. For this reason, it is logically positioned between GATT and the application layer. This position enables it to intercept and arbitrate all GATT requests while also not unduly interfering with applications or lower stack layers.

Basic Workflow. Figure 1 depicts the overall workflow when a GATT request is received from an application. When an application makes a GATT request, the platform passes the AAM a 3-tuple, consisting of the GATT request, the DeviceID corresponding to the BLE peer, and the AppID representing the requesting application. The DeviceID and AppID are assigned by the underlying platform (as described in Sect. 5.1).

The AAM separates out the three elements and queries the AAD for the DeviceID/AppID combination. The following outcomes are possible:

- **An entry exists** within the AAD for the AppID against the given DeviceID: The AAD forwards the corresponding permission value to the AAM.
  - **Stored permission is *DenyListed*:** This indicates that the user has expressly denied the application from accessing data on the specific BLE peer. The AAM indicates the deny-listed status to the platform, which should notify the requesting application that the request has failed and cannot succeed even after multiple tries.
  - **Stored permission is *Allowed*:** The AAM forwards the GATT request to ATT/GATT, receives the response, and forwards it to the platform.
- **No record exists** with the AAD for an AppID-DeviceID pair: This is signalled to the AAM, which in turn notifies the underlying platform that user authorisation is required. The user should be presented with options to allow or deny the application to access data from the BLE peer.

In this manner, only applications that are explicitly authorised by the user will be able to access data from the BLE peer, and because permissions are defined per-peer and per-application, the user has complete control over exactly which BLE devices each application can access.

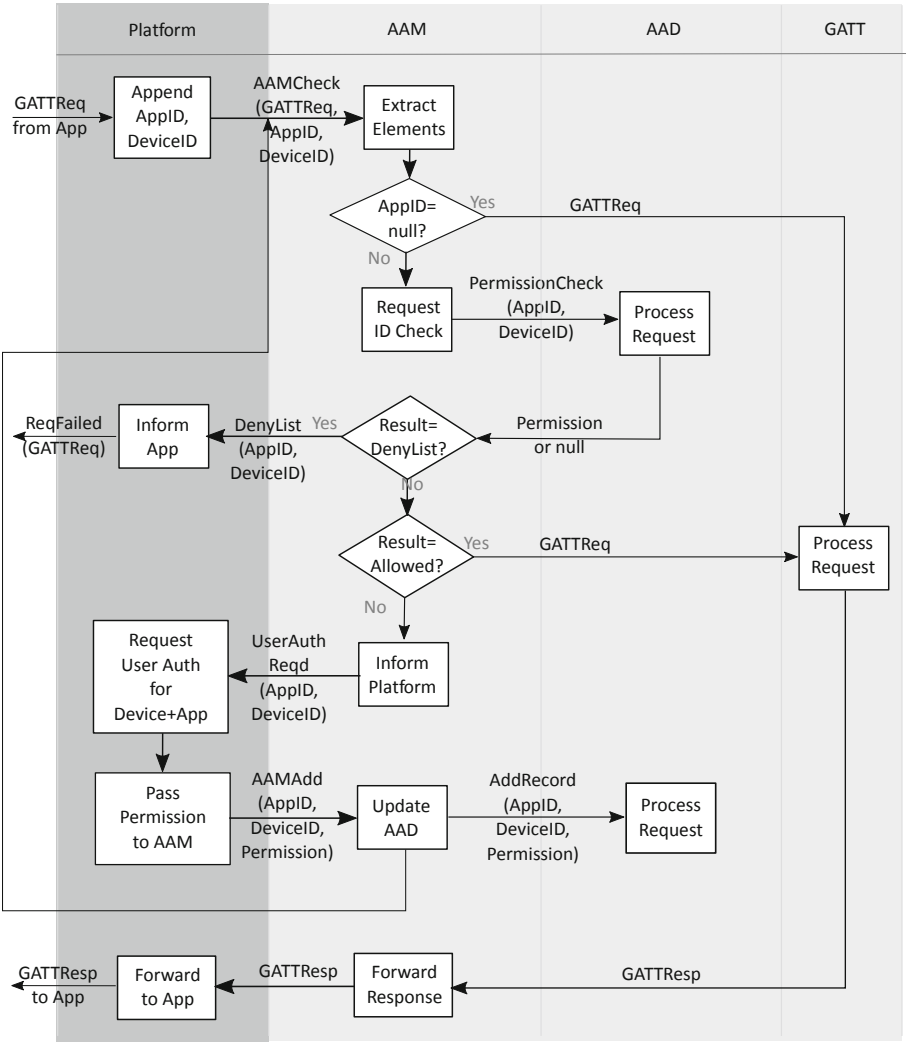


Fig. 1. Proposed workflow for GATT requests. Light grey areas are part of the BLE stack. Dark grey areas are platform/app components external to the stack.

Note also that, in this design, the functionality of GATT and other existing BLE stack components do not change. Therefore, changes to the stack are minimal. Further, the AAM only processes GATT *requests*; it forwards GATT responses as received to higher layers.

Null AppIDs. Upon receiving a `null` AppID, the AAM will forward the GATT request to ATT/GATT without any further checks. The AAM can only be sent `null` AppIDs if the underlying platform hosts a single application (see Sect. 5.3).

5.3 Device Mode

We define two modes, "Single-App" and "Multi-App", based on the application hosting configuration of the platform:

A Single-App device is a BLE-compatible device that hosts only one application. A Multi-App device is a BLE-compatible device that may host more than one application.

Examples of Single-App devices are fitness trackers, glucose monitors, door locks and insulin pumps. These devices are resource-constrained and their firmware generally contains only one set of application code. Devices such as mobile phones and personal computers, on which many applications run, would fall under the Multi-App category. The Device Mode is assigned to a device at the time of manufacture and cannot be changed during operation.

Note that the terms "Single-App device" and "Multi-App device" do not refer to how many applications a BLE-enabled device can *interface with*, but rather how many applications a BLE-enabled device *hosts*.

A Single-App device functions almost exactly as BLE does today. The AppID is set to `null`; the AAM simply passes through requests it receives from higher layers to GATT, without performing any further checks; the AAD is dormant. It is important to note that the AppID can only be `null` for Single-App devices.

A Multi-App device incorporates the complete functionality of the AAM, has a functional AAD, and behaves as described in the preceding sections. Because Multi-App devices tend to have greater storage and processing capabilities, we foresee that these changes will not be overly burdensome.

5.4 Obtaining User Authorisation

The mechanism of obtaining user authorisation will depend on the implementing platform. However, the requirement is that the mechanism must be explicitly visible to the user. We do not foresee that this will present a problem, as authorisation is only required on Multi-App platforms (such as mobile phones and laptops), which typically have fully-fledged input-output capabilities, unlike some resource-constrained Single-App platforms.

5.5 Access Revocation

It should always be possible for a user to revoke the access they have granted to an app, on a per-device basis. Similarly, it should be possible for a user to remove the *DenyListed* state for an application-device pair. This could be achieved in a similar manner to privacy controls on modern mobile and computer operating systems, where access to system resources are controlled on a per-application basis. Upon access revocation or state change, a command must be sent to the AAM, to notify the AAD to update the relevant record.

6 Requirements Analysis

In this section, we evaluate our proposed solution design against the requirements defined in Sect. 3.

SecRQ1: Prevention of Unauthorised Access to BLE Data. The AAM intercepts and processes all GATT requests from all applications on the platform. As long as the assumptions stated in Sect. 3.1 hold, no application will be able to circumvent the AAM checks and covertly access data from BLE peer devices.

SecRQ2: Per-Device Access Control. AAM checks are performed per-app, per-device. An app that has been authorised to access data from one BLE device will fail AAM checks if it has not been granted access to a different BLE device.

SecRQ3: Access Revocation. Explicit mechanisms exist within the AAM (as discussed in Sect. 5.5) to revoke access for any application that has previously been granted GATT access to a BLE device.

SysRQ1: Protection by Default. Because our solution involves the modification of the BLE specification itself (rather than of a single device, platform or application), every platform that is qualified against this design would incorporate the GATT access control mechanism, ensuring protection by default.

SysRQ2: Backward Compatibility. All new functionality in our design occurs locally, within a single device. The functionality of GATT and lower layers of the BLE stack operate as they have previously, and interface with BLE peers with no changes. Therefore, a device that implements this solution will be backward compatible with all existing BLE systems. We demonstrate this with our POC in Sect. 8, where a modified Android stack operates with an unmodified BLE peripheral. The changes also do not affect the existing Bluetooth services or profiles.

SysRQ3: Minimal Overhead for Resource-Constrained Devices. The processing described in Sect. 5.2 applies to Multi-App devices such as mobile phones, which are expected to have reasonably powerful operating systems and fewer restrictions in terms of battery usage. Most BLE peripherals have limited storage and do not support hosting multiple apps. Therefore, such devices will be defined as Single-App, and will be spared most of the processing overhead (Sect. 5.3).

7 Additional Benefits

In this section, we describe advantages of our proposed solution, in addition to those implied by the fulfilment of the requirements.

No Changes to Existing BLE Stack Layers. In our solution, a single new layer is added to the stack, and it is within this layer that the bulk of the access control behaviour is implemented. No modifications are required for any of the existing BLE stack layers, including the ATT/GATT layer, which is the only core layer that interfaces with the AAM (i.e., the requests received by GATT with the proposed new stack will be exactly as they are at present). This makes it easier for the Multi-App platform to implement the changes in a modular fashion (assuming the existing stack has also been developed in a similar manner).

No Changes to Applications. Our proposed solution requires no interaction between an application and the AAM. This means that applications that issue GATT requests will not require any changes, apart from possibly to handle a new error status. This is a significant advantage, as there are several thousand mobile applications with BLE capabilities in existence today [25,32], and making changes to all of them would be extremely challenging, as it would require cooperation from a large number of developers.

Equal Protection for All Services. A BLE device may implement services defined by the Bluetooth SIG, but may also implement its own custom services. As with the example provided in Sect. 2, most services and profiles defined by the SIG, including those that read user health data such as heart rate or glucose measurements, do not specify higher-layer protection as a security requirement. With our solution, protection is applied to both types of services, even if SIG-defined services do not specify authorisation permissions.

Protection Even in the Absence of Pairing. Many BLE peripherals tend not to have sufficient input-output capabilities, and therefore either implement weak pairing or no pairing at all [26]. Our solution is separate from and at a higher layer to possible link layer protection mechanisms such as pairing. Any GATT request from an application has to first pass through the AAM before it can be forwarded via the link layer to the BLE peer, which means that protection is applied at a much earlier step. This means that the proposed new stack will protect data on a BLE peer from access by an unauthorised application on a Multi-App platform even if the peripheral does not specify a requirement for pairing. Of course, if the peripheral does not require pairing, then its data can be eavesdropped over the wireless interface; it can also be accessed by a different unauthorised device. However, that is outside the scope of this work. Our work focuses on protecting data from unauthorised access *at the application layer*.

Most Changes Are to Mature Platforms. While a specification change would typically require a change to all devices that implement the BLE stack, the way in which the proposed change has been designed allows for the system to function even without any changes to existing peripherals. The only entities that will *require* changes are Multi-App platforms such as mobile or personal

computer operating systems. These platforms tend to have a robust update mechanism in place already, which is familiar to users. This ensures greater likelihood of the changes actually being installed on end user devices.

Potential for Fine-Grained Access Control. Our current solution design enables an application to either access all data on a peripheral or none. This can be extended to enable fine-grained access control by access type (i.e., reads or writes) or even on a per-characteristic level (we note, however, that per-characteristic access control is inadvisable in most cases, as it would place too much decision burden on users who may not be aware of the purpose of each characteristic).

8 Proof of Concept

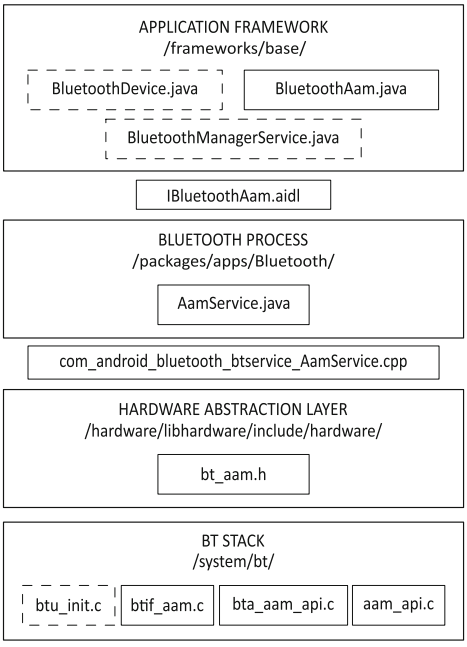


Fig. 2. Main changes made to Android architecture for POC. Utility functions not shown. Solid & dashed lines denote new & modified components, respectively.

In order to demonstrate the viability of our proposed solution, we have implemented a Proof-of-Concept on the Android-x86 platform.[3]
In this section, we discuss implementation details, describe the test set-up, and evaluate the POC in terms of development effort, performance overheads and user experience.

8.1 Implementation Details

We selected the Android platform for our POC due to its open-source nature, large installation base and potential familiarity to readers. We used the Android-x86[4] code base, to be able to implement and test our solution on a virtual machine, without the need for expensive device installations.[5] The modified Android-x86 was built on a VM running Ubuntu 18.04.3 LTS with 128 GB RAM (8 GB for heap) and 8 cores.

[3] https://github.com/projectbtle/BLE-MultiApp-POC.
[4] Android-x86 is a port of Android for x86 platforms. It is based on the Android Open Source Project (AOSP) with some modifications.
[5] Android-x86 offers Bluetooth capabilities, which official Android emulators lack.

Figure 2 depicts the components within the Android-x86 framework that were modified or added for our proposed solution. Specifically, on Android, a GATT request (such as those for reading or writing characteristics) can only be issued after an app has called the `connectGatt` method. Because of this construct, and due to the nature of the Android architecture, we select `connectGatt` as the entry point for the AAM checks. We use the device's hardware address as the DeviceID and extract the Android app's application ID (which uniquely identifies an app on the Android platform [2]) to use as the AppID.

The actual AAM functionality is implemented within the BT stack. In keeping with Android's workflow for other BLE functionality, we implemented the AAM functionality along a path from the application framework to a custom AAM "layer" within the stack, as shown in Fig. 2. Within the AAM layer, the AAD is implemented as a linked-list of records, following the same structure that is used natively by Android for storing pairing credentials.

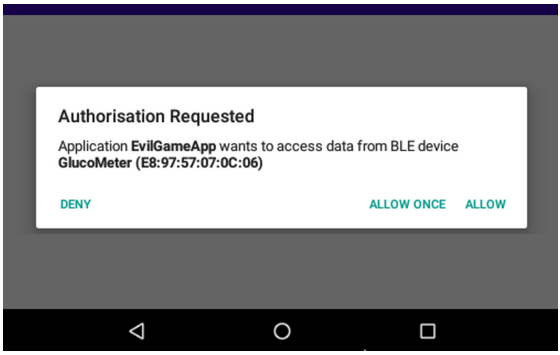


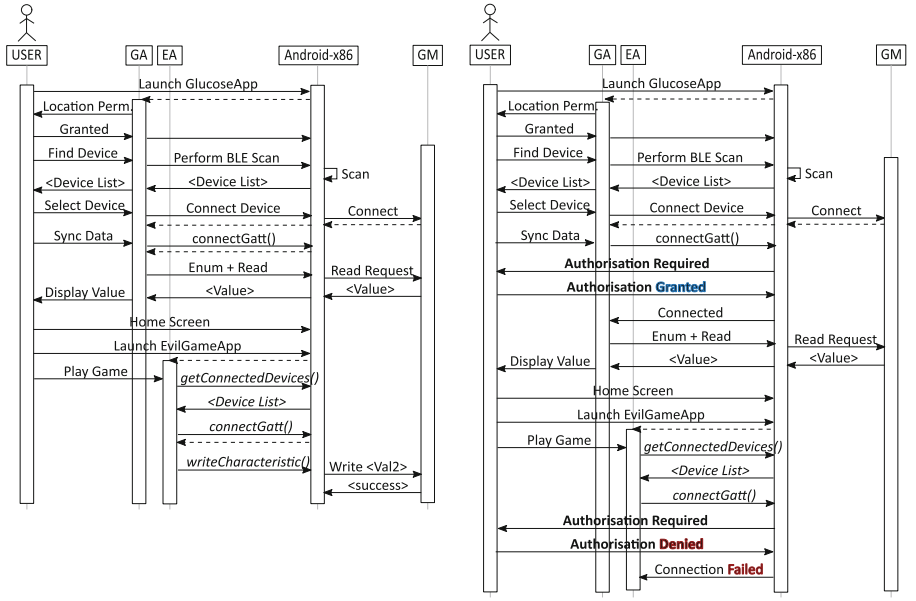
Fig. 3. User authorisation dialog with explicit reference to app and BLE device.

User authorisation is requested via standard Android dialog boxes. To make the contents of the dialog box clear and easily understood by the user, the application name is displayed instead of the application ID. For the device, both the device name (if available) and the hardware address are displayed, to avoid ambiguity in situations where more than one device with the same name are advertising in the vicinity. A sample is depicted in Fig. 3. The `Allow` and `Deny` options within the dialog box map to the *Allowed* and *DenyListed* permissions described in Sect. 5.1, respectively. We have also implemented a temporary access option *AuthReqd*, which is displayed as `Allow Once` in the dialog box.

8.2 POC Tests

To test our POC, we replicate the attack scenario described in [25]. The attack demonstrates covert access of data from a connected BLE device by an unauthorised app. We utilise four main components:

1. VMWare Workstation 14 Player on a Windows 10 laptop, with a CSR adapter, for running the original and modified Android-x86 builds.
2. Nordic nRF51 DK, in the role of a glucose meter ("GlucoMeter"). No pairing-protected characteristics.
3. Android app, in the role of a glucose monitoring app ("GlucoseApp").
4. Android app, in the role of a malicious app masquerading as a legitimate app, e.g., a game, which accesses BLE data covertly ("EvilGameApp").



(a) Current scenario. (b) With our solution.

Fig. 4. Interaction between User, GlucoseApp (GA), EvilGameApp (EA), Multi-App platform (Android-x86) and BLE GlucoMeter (GM). Items in *italics* are interactions between EA and Android-x86 that occur without user awareness. Items in **bold** are new user interaction elements.

We deploy a VM with the original Android-x86 build and perform the following functions in order: (i) Launch GlucoseApp. (ii) Scan for BLE devices. (iii) Connect to the GATT server on the "GlucoMeter" and read a characteristic. This will read a dummy value of 0x12345678. (iv) Launch EvilGameApp (which covertly identifies the existing connection to the GlucoMeter, calls `connectGatt` to it and writes the same characteristic). Figure 4a depicts the interactions between five main entities (the user, GlucoseApp, EvilGameApp, the Multi-App platform (i.e., Android-x86), and the BLE device) when going through the above test steps in the absence of any protection mechanism. We then repeat the tests using the modified Android-x86 build. Figure 4b illustrates the interaction between the five entities when our controls have been implemented. The two figures demonstrate that unauthorised data access is prevented with our solution because the covert data access attempt is brought to the attention of the user and defeated, i.e., protection is achieved due to explicit user awareness.

Testing with Pairing-Protected Data. We additionally modified the "GlucoMeter" to require pairing prior to data access. Re-running the tests again, we found that our controls worked in that scenario as well, as expected.

Testing with Real-World Devices and Applications. We verified the functionality of the POC implementation on real-world devices and apps by testing two popular fitness trackers, the Mi Band 2 and ID107HR/VeryFit, against the modified Android-x86. For this, we installed the corresponding applications on the POC platform and connected to the devices. The solution worked without the need for any modifications to the fitness trackers or to their apps.

8.3 Evaluation

Development Effort. The entire set of modifications in our POC, including substantial debugging information, required approximately 1500 new lines of code. This demonstrates that the solution is viable.

Performance Overheads. Analysing Android debug logs, we identified that performing an AAM access check took at most 25 milliseconds. This is well within the 100-millisecond instantaneous reaction perception limit [12,19]. We found while interacting with the system that this amount of time is indiscernible (from our point of view as a user).

User Experience and Comprehension. The user is shown a dialog when an application is first launched and attempts to access data from a BLE device. Once that dialog has been responded to, subsequent access attempts don't require user interaction. Therefore, it is the impact of the first dialog that needs to be analysed. Due to the prevailing Covid-19 situation, we were unable to conduct in-person tests. We therefore present here a theoretical analysis of the impact on user experience and comprehension.

If a malicious app professes to be benign but covertly accesses BLE data, then it may limit the number of permissions that it requests in order to trick the user into believing that it is harmless. In such a scenario, the presentation of a system dialog could serve to call the user's attention to the fact that covert data access is being attempted. Previous studies on user authorisation mechanisms [17], such as the ones used in the Android permission system, suggest that using a system dialog the first time a resource is accessed provides the optimal point for user decision making. Our proposal effectively achieves this by raising the authorisation dialog the first time a BLE resource is accessed. If the access to that resource (a BLE peripheral in our case) is malicious, the user won't see how the app functionality is related to the device access and will therefore deny it. Of course, if the malicious app portrays itself as a BLE accessory app, it is more likely for the user to allow access. Identifying malicious behaviour *after* access to the BLE peripheral has been granted (e.g., leakage of BLE data once it has been read from a device) is outside the current scope and is left as future work.

9 Limitations

Use of External Sources of Information. The proposed solution requires that the implementing platform supply unique application identifiers to a component within the BLE stack. This in effect removes the self-contained aspect of the stack by introducing an element external to the stack.

Reliance on Honesty of Platform. While the proposed solution enables the user to grant permissions on a per-device, per-app basis, the fact that the access control checks are entirely performed by the Multi-App platform implies that there is implicit reliance on the integrity and honesty of the platform. That is, there is an underlying assumption that the Multi-App platform will apply access control checks to all applications in an unbiased manner. However, it may be the case that a Multi-App platform which ships with its own set of apps may automatically authorise those apps to access any BLE peer, and only apply access control checks to third-party apps. This would then remove some of the visibility and control from the user. Circumventing such an issue would require a complex protocol between the platform and peripheral, and is outside the scope of this paper. For our work, we make the assumption of an honest and fair platform.

Complexities in Desktop/Laptop Environments. The solution we have proposed is straightforward to implement on mobile operating systems, where the level of user customisation, particularly regarding the BLE stack, is minimal. However, with operating systems such as Windows and Linux, there is a possibility that an application might use a BLE stack that is not provided by the OS. For example, it is possible on a Windows machine to utilise an external BLE adapter, instead of the system-provided Bluetooth capabilities. This is done manually, with additional hardware, and requires that the system Bluetooth be turned off. While such a setup is more likely to exist in testing scenarios than in normal user systems, it is still a consideration that should be factored in when implementing the solution.

10 Related Work

Various aspects regarding the security and privacy of Bluetooth Low Energy have been studied over the years. The BLE pairing process has come under specific scrutiny, with passive eavesdropping [10,23], authentication bypass [22], key entropy downgrade [3] and link encryption downgrade vulnerabilities [29,31] explored over time. Man-in-the-Middle/spoofing attacks have been described in [14,29]. Privacy concerns with BLE have also been widely studied [8,9,13].

The works most closely aligned with ours are [25] (which is our own work, where we first describe the unauthorised data access vulnerability for BLE) and [18]. In [18], the authors described unauthorised data access for Bluetooth BR/EDR (i.e., "Classic") devices by any Android app with Bluetooth permissions. They proposed an Android OS-level protection mechanism via bonding policies. Their solution assumes that the first app that pairs to a Bluetooth device is the authorised app and automatically creates a policy. Our solution makes no such assumption, particularly since, in the BLE case, the peripheral device may not require pairing at all. Further, the user may desire the use of a secondary application with additional features. Our solution instead explicitly informs the user of any application that makes a GATT request to a connected BLE device. This ensures that the user is aware of and can make decisions regarding whether or not to allow access. We also present our solution as a specification-level change, which then affords protection by default for the entire BLE ecosystem.

11 Conclusion

We have presented a modified Bluetooth Low Energy stack to solve the unauthorised data access vulnerability on multi-app platforms. Our solution fulfils stringent security and system requirements, and takes into account practical considerations in its design. It ensures protection by default, while maintaining backward compatibility with existing systems. No changes are required to apps or resource-constrained BLE peripherals, nor are changes required to existing stack layers. We have also implemented a proof-of-concept on the Android-x86 platform to illustrate our solution, and have demonstrated that the solution prevents unauthorised data access via explicit user awareness and authorisation.

References

1. Android: Bluetooth Low Energy overview (2020). https://developer.android.com/guide/topics/connectivity/bluetooth-le. Accessed 06 Feb 2021
2. Android: Set the application ID (2020). https://developer.android.com/studio/build/application-id. Accessed 20 Oct 2020
3. Antonioli, D., Tippenhauer, N.O., Rasmussen, K.: Low entropy key negotiation attacks on Bluetooth and Bluetooth Low Energy. IACR Cryptology ePrint Archive (2019)
4. Apple: If an app would like to use Bluetooth on your device (2019). https://support.apple.com/en-us/HT210578. Accessed 20 Oct 2020
5. Assal, H., Chiasson, S.: 'think secure from the beginning' a survey with software developers. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp. 1–13 (2019)
6. Bluetooth Special Interest Group: 2020 Bluetooth market update (2020). https://www.bluetooth.com/bluetooth-resources/2020-bmu. Accessed 13 May 2020
7. Bluetooth Special Interest Group: LaunchStudio (2021). https://launchstudio.bluetooth.com/Listings/Search. Accessed 08 Feb 2021

8. Celosia, G., Cunche, M.: Fingerprinting Bluetooth-Low-Energy devices based on the Generic Attribute profile. In: Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things, pp. 24–31. ACM (2019)
9. Das, A.K., Pathak, P.H., Chuah, C.N., Mohapatra, P.: Uncovering privacy leakage in BLE network traffic of wearable fitness trackers. In: Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications (2016)
10. Gomez, C., Oller, J., Paradells, J.: Overview and evaluation of Bluetooth Low Energy: An emerging low-power wireless technology. Sensors (2012)
11. Halderman, J.A.: To strengthen security, change developers' incentives. IEEE Security & Privacy (2010)
12. Hogan, L.C.: Performance is user experience (2014). https://designingforperformance.com/performance-is-ux. Accessed 15 Feb 2021
13. Issoufaly, T., Tournoux, P.U.: BLEB: Bluetooth Low Energy Botnet for large scale individual tracking. In: 2017 1st International Conference on Next Generation Computing Applications (NextComp), pp. 115–120. IEEE (2017)
14. Jasek, S.: Gattacking Bluetooth Smart devices. In: Black Hat USA (2016)
15. van der Linden, D., et al.: Schrödinger's security: opening the box on app developers' security rationale. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (2020)
16. Markets and Markets: IoT chip market (2021). https://www.marketsandmarkets.com/Market-Reports/iot-chip-market-236473142.html. Accessed 08 Feb 2021
17. Micinski, K., Votipka, D., Stevens, R., Kofinas, N., Mazurek, M.L., Foster, J.S.: User interactions and permission use on Android. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 362–373 (2017)
18. Naveed, M., Zhou, X., Demetriou, S., Wang, X., Gunter, C.A.: Inside job: Understanding and mitigating the threat of external device mis-binding on Android. In: 21st Annual Network and Distributed System Security Symposium (2014)
19. Nielsen, J.: Response times: The 3 important limits (1993). https://www.nngroup.com/articles/response-times-3-important-limits. Accessed 15 Feb 2021
20. O'Donnell, L.: Consumers urged to junk insecure IoT devices. https://threatpost.com/consumers-urged-to-junk-insecure-iot-devices/145800. Accessed 12 Feb 2021
21. Ramachandran, R.: IoT connected healthcare devices: Challenges in cybersecurity and the way forward (2020)
22. Rosa, T.: Bypassing passkey authentication in Bluetooth Low Energy. IACR Cryptology ePrint Archive (2013)
23. Ryan, M.: Bluetooth: With low energy comes low security. In: 7th USENIX Workshop on Offensive Technologies (2013)
24. Schwartau, W.: Let's end pass-the-buck security (2004)
25. Sivakumaran, P., Blasco, J.: A study of the feasibility of co-located app attacks against BLE and a large-scale analysis of the current application-layer security landscape. In: 28th USENIX Security Symposium, pp. 1–18 (2019)
26. Sivakumaran, P., Blasco Alis, J.: A low energy profile: analysing characteristic security on BLE peripherals. In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, pp. 152–154. ACM (2018)
27. statcounter: Operating system market share worldwide. https://gs.statcounter.com/os-market-share. Accessed 06 Feb 2021
28. Woolley, M.: Bluetooth 5 (2019). https://www.bluetooth.com/wp-content/uploads/2019/03/Bluetooth_5-FINAL.pdf. Accessed 31 Aug 2020
29. Wu, J., et al.: BLESA: spoofing attacks against reconnections in Bluetooth Low Energy. In: 14th USENIX Workshop on Offensive Technologies (2020)

30. Zandberg, K., Schleiser, K., Acosta, F., Tschofenig, H., Baccelli, E.: Secure firmware updates for constrained IoT devices using open standards: a reality check. IEEE Access (2019)
31. Zhang, Y., Weng, J., Dey, R., Jin, Y., Lin, Z., Fu, X.: On the (in) security of Bluetooth Low Energy one-way Secure Connections Only mode (2019)
32. Zuo, C., Wen, H., Lin, Z., Zhang, Y.: 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 (2019)