Bluetooth Beacons and How to Make Users Untraceable

Yanyan Ren
Brown University

Yuchen Yang
Brown University

Abstract

Bluetooth beacons provide a useful way to interact with phone users based on their locations, but this also means that beacons can track users at a very fine-grained level. We come up with a solution to prevent users from being tracked while still getting the service. Our demo contains four parts: an Android phone application, a blackbox server that masks user's identity, a registration site that records a beacon's corresponding server address, and a server that sends coupon code to users. We believe that our demo shows a new way for phone application to interact with beacons without compromising the user's privacy.

1 Introduction

Bluetooth beacons are cheap and lightweight devices to gather phone's location information. However, your smart phone is tracking you using these beacons. As the beacons are cheap and they are everywhere, it is inevitable that mobile applications can use them to get your location. The issue is, some sneaky applications are tracking you but you do not even know about it and your location and habits are already sent to some third-party people.

The problem is exacerbated by two issues. First, beacons are cheap and low-energy which means using them cost little. Previously, Facebook even distribute free beacons to businesses for location marketing inside the Facebook app. [6] However as normal users, we cannot find out what application is using the information. Second, applications do not need much permission to do the monitoring work. For many of mobile phone users, bluetooth is on all the time. Smartphones running Google's Android software may even collect your location data even with Bluetooth off. [7]

We hope to provide some solutions to the problem while the beacons still function as normal. We wish that the servers would not know our identity but it still pushes the notifications or coupons to our phone. So the straightforward solution would be that the phone send the packet to a middlebox, which modifies the device information and forwards the packet to the server while pretending itself as a phone. As there are so many applications using beacons, it is impossible to capture all the mobile traffic and parse the packet format for every application. So the only possible way is to design a standard packet format for all Beacon-related traffic and create a blackbox to forward the packets without carrying any device information.

Thus, we present a brandnew beacon anonymization infrastructure, which can hide our identity from beacon owners' notification servers. The infrastructure lets the phone to send the beacon UUID to the blackbox, which first consults an official registration site for the notification server URL and then communicates directly to the notification server. For businesses who want to provide services using beacon but do not want customers to worry about privacy, our infrastructure fulfills their goal and saves their time of developing and maintaining a separate application. All they need to do is deploying the beacon, registering on an official cite and providing a server.

2 Background

Bluetooth Beacons are devices that continuously send Bluetooth Low Energy(BLE) signals that cover the range of about 80 meters [3]. The signals can be received by nearby mobile devices and then processed by the mobile applications. The accuracy of localization is about 5 meters. Note that the GPS has a similar accuracy of localization, but it doesn't work well indoors [6]. Since the beacon only needs to broadcast a small amount of information (mostly UUID), the device can be quite small and use little energy. It is often used by companies to track customers inside stores.

There are mainly 2 kinds of protocols used for bluetooth beacons.

iBeacon [5] is developed by Apple and is the most popular protocol used in the area. It does better jobs for interactivity between iOS devices and iBeacon hardware. For beacons using iBeacon, they have better integrity at identifying bea-

cons and applications can direct the information of beacons to iBeacon servers. Universally unique identifier(UUID) is the only information needed to find the developer server.

Eddystone [1] is developed by Google and is a more flexible one that contains security features and APIs but requires more knowledge from developers [2]. The broadcasted packets not only contain the UUID, but also sensor telemetry(temperature, battery status) and a URL address of the developer's server. This indicates that it does not require specific applications from the developer and can get the response directly from a common beacon application.

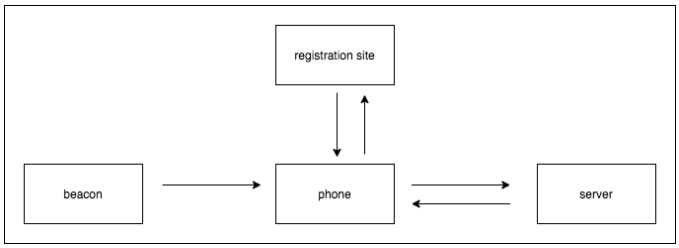

Figure 1: How a beacon works

Since beacons have a very simple structure and uses low energy, the packets sent usually contain little information such as just the UUID. On the registration site, more information can be stored alongside with the UUID, such as latitude and longitude, indoor floor level and room number, text description, and any arbitrary property as key/value pairs.

Figure 1 shows how the beacon, the phone, registration site, and the server interact with each other. The beacon constantly emits packets containing its UUID to nearby phones. When some app in a phone recieves a packet from the beacon, it sends the beacon's UUID to the registration site and gets back the beacon's corresponding information, such as the location information that reveals the user is currently standing in front of a yogurt counter. Another piece of the information would be the store's server address. The phone then talks to the server, sending the location information obtained from the beacon registration site and perhaps some personal information like email address stored in the app. The server sends back appropriate messages back to the phone, such as a yogurt coupon of a brand that the user has purchased last week.

3 Design

We want to design a spoofing utility so that a user can be untraceble on the server's side. We want to keep the general functionality of the beacon and the application, but doesn't want to expose the user's personal information. Using the yogurt coupon example mentioned before, the user should still get a coupon (and instead of the user's favorite brand, it's a not so popular brand that needs promotion), and the server wouldn't get any more information than someone is in front of this yogurt counter.

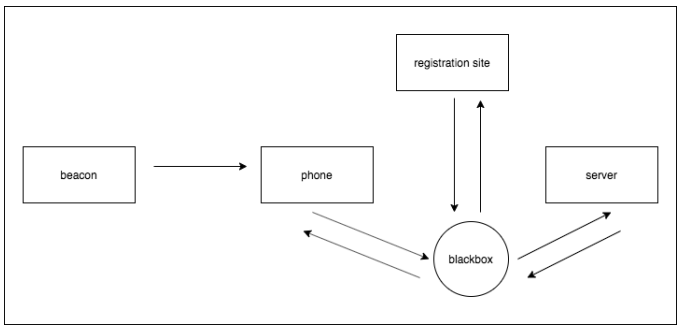

Figure 2: Spoofing with our blackbox

To make this spoofing utility, we put a blackbox between the phone and the server's communication 2. With our blackbox in the middle, the phone gets mapped to a temporary id before talking to the server. Thus, the server can't see any personal information about the phone, but can still responds according to the beacon's information attached. When the phone recieves a packet from a beacon, the phone forwards the packt to the blackbox. The blackbox looks up beacon's information on the registration site, and talks to the server accordingly. When a packet is sent back from the server, our blackbox matches the temporary id to the phone, and delivers the packet.

At first, we looked for existing application that pass beacon information to a server, and planned to capture the packets exchanged between the phone and the server. However, all the application that we could find only use the beacon information within the phone (e.g. attach the beacon to your wallet, and the phone would make a noise if it gets too far from the wallet/beacon). This difficulty was expected, since it's in the applications' best interests to hide from the user that they are passing the beacon information to their server, so that they can keep collecting informtation quietly in the background.

We then thought about capturing all packets leaving from the phone. The end product would be a gatekeeper application that once turned on, would filter out all the beacon related packets and forward them to the blackbox. This approach faced two challenges. First is decrypting the packets, and this is especially hard for HTTPS packets. Second is efficiency, since we don't want the other packets get blocked at our gatekeeper app.

Therefore, we build our own application, registration site and server so that we can have total control of how they interact with the blackbox. Inside the application, we make sure that when the application detects a beacon, it opens a socket to forward the beacon's UUID to the blackbox, and keeps the socket open until it gets back a coupon code. For a more generalized use, our approach requires the collaboration from the application developer – the application needs to talk to the blackbox instead of going straight to the server.

The big limitation here is that our approach won't stop any

of the existing application from quietly forwarding the beacon information and personal information to the server. However, for the application that actively use the beacon information to interact with the users, we believe adopting our approach may make the application more appealing to the users. In the yogurt coupon example, the user might get freaked out by how good the application is at knowing their favorite brand, and decide to delete the application because of this. But if the application developer uses our blackbox approach, and explains to the user that their personal information is safe and they are just getting more coupons, the users might be more likely to keep and use the app.

4 Implementation

The current implementation is pretty simple. It is separated into four components. We have an Android side application, a blackbox, a registration site and several notification servers.

"Poll()" is a system call to wait for some events on a file descriptor. It is the technique used in three of the four components. As servers, all of the blackbox, the registration site and the notification servers need to handle multiple connections from clients at the same time. Poll waits for one of a set of file descriptors to become ready to perform I/O. It provides the ability to keep waiting for new incoming connections on one file descriptor and maintain the established connection with other file descriptors. In our case, there is a file descriptor list and the first is the one receiving new connections and the others are responsible for existing connections.

Application code contains two parts, a beacon scanner and a client of the blackbox-Android connection. For the beacon scanner, we directly borrow the code of an existing beacon scanner [4]. The application uses Kotlin and can display important information for all types of beacons. In our case, we focus on iBeacon so it displays the UUID, Major, Minor, RSSI, MAC address, battery level and some other status information. The client is a simple socket connection in Kotlin. We build the client with a reader and a writer, which sends just the UUID information and receives the response afterwards. Once the client gets the response, we directly close the connection on the client side.

Blackbox is the one who talks to all the other components. Thus we need to setup three kinds of connections, to the phone, to the registration site and to notification servers, separately. It uses "poll()" to handle multiple traffic from mobile clients. After getting the UUID from a client, the blackbox holds the connection and talks to both the registration site and a notification server. Getting the notification server from registration site using the UUID, the blackbox is able to communicate with the notification server. The response from the notification server is directly sent back to the mobile client and the connection is closed. The complete process is shown in Algorithm 1.

Registration site maintains a table of the UUIDs and the corresponding server URLs. Before the beacon is in use, people come to the site to register the beacon ID and the server URLs. After receiving the UUID from a blackbox server, it checks the table and get the server URL. In our demo, the table lists the server IP and port number, so the response is in the format of "IP:port". All the information is initialized and hardcoded with the IDs of the beacons we use in the demo. It also uses "poll()" to handle the traffic from multiple blackboxes.

Algorithm 1 Blackbox server

```
pollfds ← new file descriptor array
pollfds[0] ← POLLIN
while Not timeout do
    rc ← poll(pollfds, num_fds, timeout)
    for i do in pollfds
        if pollfds[i].revents == POLLIN then
            if pollfds[i].fd == 0 then
                pollfds[i] ← acceptconnection
            else
                receive message
                consult registration site
                communicate with notification server
                send notifcation back through pollfds[i]
```

Notification servers just receive pings and respond with notifications or coupon codes. In our case, we generate a random 40-bit string along with some texts as the response. Similarly, poll is used for handling multiple blackbox connections.

5 Evaluation and limitation

We have developed a prototype of the beacon anonymization system. The focus of our evaluation is to check whether the notification servers can know the behavior of a single device and whether the system can handle multiple mobile devices and blackboxes at the same time. To this end, we use multiple machines and let them pretend to be separated phones. Then we send UUIDs from the phones and see how well the system works. Every notification server also maintains a list of IPs which visited the site, to see whether any device information is shown at the server end.

Our system still contains several limitations.

1. The users need to trust the blackbox, or users need to use the application code we provide. We still need to add some privacy system on the blackbox and secure protocols for inter-component traffic.

2. The system does not work for separate mobile applications which delibrately collect the bluetooth information. It only works for beacons which only has a notification server and is registered on an official registration cite.

3. We need more blackbox servers to make the traffic sparser. Otherwise the blackboxes would be easily classified as DDOS attacker.

References

[1] Beacons | google developers. https://developers.google.com/beacons.

[2] ibeacon vs eddystone: Which one works better for your pilot project? https://blog.beaconstac.com/2016/01/ibeacon-vs-eddystone/.

[3] Beaconstac. What is a bluetooth beacon? how do ble beacons work? https://www.beaconstac.com/what-is-a-bluetooth-beacon#working.

[4] Bridouille. Android beacon scanner, Dec 2018. https://github.com/Bridouille/android-beacon-scanner.

[5] Apple Inc. ibeacon. https://developer.apple.com/ibeacon/.

[6] Michael Kwet. In stores, secret bluetooth surveillance tracks your every move, Jun 2019. https://www.nytimes.com/interactive/2019/06/14/opinion/bluetooth-wireless-tracking-privacy.html.

[7] David Yanofsky. Google can still use bluetooth to track your android phone when bluetooth is turned off, Jan 2018. https://qz.com/1169760/phone-data/.