Reverse Engineering a Bluetooth API

Maxwell Turpin
Mentor: Ming Chow
December 14, 2016

Web App and Source Code

Web app: https://mhturpin.github.io/94fifty/
Source code: https://github.com/mhturpin/94fifty/

Abstract

Bluetooth devices have become so prevalent that they can be seen practically everywhere, from wireless headsets and earbuds to household items like shades and lightbulbs. As they spread further into our lives, it is important that we understand any security risks they might be bringing with them. If someone were to gain unauthorized control of the device, they could cause some serious damage, from stealing information to even killing someone, if they were controlling a medical device like an insulin pump. Because of these potential security holes, it is important to test the devices thoroughly. However, the apps released by companies to control their devices are often limited in the functions they provide, and don't allow for much thorough testing. Therefore, it is necessary to a tool that gives more control to the tester. In this paper, we will be detailing how to reverse engineer the API for a Bluetooth basketball and how it can be applied to other Bluetooth devices.

Introduction

As Bluetooth devices have become increasingly integrated into our lives, and especially as with the recent release of Bluetooth 5 bringing more Internet of Things (IoT) features [1], the need for securing such connections is even more crucial. The IoT is the collection of all the devices that connect to the internet. As we saw this past October with the DDoS attack on Dyn, unsecured devices on the IoT are vulnerable to attack, and beyond the dangers to the user that come with those being compromised, they can be used in a botnet to carry out even larger attacks that can take out large portions of the internet [2].

As is the case with a significant amount of our technology, development is focused mainly on adding features, with security coming as an afterthought. Since the apps to control these Bluetooth devices are often limited in the control they provide, it is difficult for anyone to investigate the security and discover existing flaws. Because many Bluetooth devices do not have their communications secured or encrypted, anyone who knows the proper format can connect to and control that device, or simply record the data being sent to and from it. While this itself is a security flaw, we can use this to our advantage in reverse engineering an app to control the device.

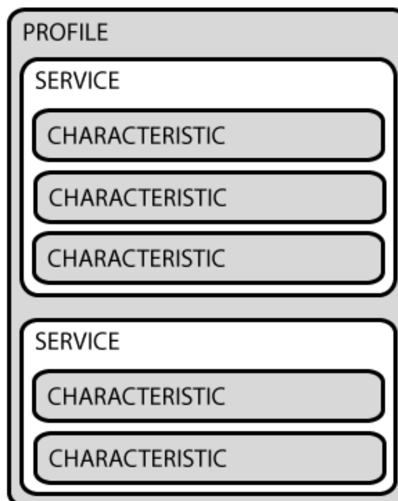
To the Community

I chose this project because I like deciphering and understanding the underlying mechanisms to how things work. Reverse engineering incorporates a lot of this, because you need to dig into the source code as well as look data to put together how a piece of software works. When Ming announced that he had a basketball he wanted reverse engineered, I realized that it was the perfect project for me.

Why should you care? Although you may never have thought about it, the IoT is everywhere and its reach is expanding rapidly. Bluetooth 5 just released, and introduces many features that allow it to be used with IoT devices. Everything from your Fitbit to your smart light bulbs, to your smart thermostat are connected, and therefore vulnerable to attack. If these aren't secured properly, someone could, for example, put ransomware on your thermostat that takes control of your house until you pay them off. Although security isn't always visible, it's as important, if not more important, than any other features.

Bluetooth Background

Bluetooth is a protocol that allows two devices to communicate easily with one another. It does this by sending packets of data between them. The way these devices communicate is through the Generic Attribute Profile (GATT). The communication system is broken down into a hierarchy of Profiles → Services → Characteristics. Characteristics are the most basic and allow you to send data of a single type, for example setting the color of a light bulb. Services group similar characteristics together, for example all different light bulb setting controls. Finally, a profile is a collection of those services. When an app communicates with a Bluetooth device, it sends data to a characteristic, which the devices receives and interprets.



[3]

Reverse Engineering Process

I used the process outlined in [4] to guide me. The links to the web app and source code I created are listed at the beginning of this paper.

nRF Connect

nRF Connect is an app which allows you to view all the Bluetooth devices broadcasting near you. You are also able to filter if you are looking for a specific device. Once you see your device, you can connect to it and view available services and characteristics. With the basketball, I saw that it has a service numbered 'b69bc590-59d9-4920-9552-defcc31651fe'. I was not able to see any characteristics belonging to that service though, but we'll come back to that when I discuss the web app. In some cases, if the interface is simple enough, it is possible to send random values to a characteristic and deduce the format. With the case of the basketball, this wasn't possible, so I moved on to the next step of recording the Bluetooth traffic.

Bluetooth HCI Snoop Log

On Android devices under 'Developer options', there is a setting to 'Enable Bluetooth HCI snoop log'. Once I turned this on, I went to the 94Fifty app, connected the basketball, and did a quick activity. With the Android SDK installed, I connected my phone to my computer, opened the terminal, and ran the command:

```
adb pull /sdcard/btsnoop_hci.log
```

This saves the file to your current directory. Once I had the file, I opened it in Wireshark. To view only the packets involving the basketball, I found its MAC address (bc:6a:29:2e:b3:75) in one of the packets and filtered the packets with:

```
bluetooth.addr==bc:6a:29:2e:b3:75
```

I noticed that the majority of packets being sent began and ended with the hex value 0x7e, but beyond that it was impossible to tell what the contents meant. Therefore, I moved on to the next step of decompiling the APK.

APK Decompilation

I took the Google Play Store URL of the 94Fifty app and used website [5] to download a copy of the APK. I then uploaded that to [6], from which I downloaded a decompiled version as Java source code. The code for creating and interpreting packets is in 'com/_94fifty/protocol/v1'. From that I was able to find a switch statement listing the type code that comes at the beginning of every data segment. Each type has a class that can be referenced to see how it creates its packet. The general format of a packet is:
Start byte (0x7e), type, token, [all necessary data], Crc32 checksum, end byte (0x7e).

I used this information to write a script that takes the data sent from the basketball doing a dribble activity, and decodes it into its component data elements. Although I figured

out most of the data, I couldn't find where or how the token was created. I believe it is generated fresh with each connection, but beyond that I'm not sure. I think it is also important to sending valid packets to the ball.

Creating a Web App

Using the information obtained above, I attempted to create an app that would allow the user to control the dribble activity. I got it to the point where I could send data to the ball, but because I wasn't able to completely decode how the information in the sent packets is generated, I couldn't create a valid packet. I will; however, describe my process up to that point, as I believe if a proper packet is created, controlling the ball with this app would be possible.

For the most part, I followed the instructions outlined in [7], so I will only discuss points where I deviated from them. The source code is located at [8].

1. I signed up for an Origin Trial token to enable web Bluetooth, which takes about a day, so to use the site while I was waiting I enabled the web Bluetooth flag under 'chrome://flags'.
2. I copied the sample code onto the website, but changed 'filters' under 'navigator.bluetooth.requestDevice' to '[{ namePrefix: ['B'] }]' since the basketball's name is Basketball. I also added 'optionalServices: ['b69bc590-59d9-4920-9552-defcc31651fe']' so that I could connect to that service later on.
3. After ensuring that the website could recognize the ball, I needed to determine what characteristics the service had, since I was unable to find them in nRF Connect. To do this I added a slightly modified version of the code in [9] so that I could print out all characteristics. I found that it had two:
   One was '8b00ace7-eb0b-49b0-bbe9-9aee0a26e1a3' of type Write Without Response
   The other was '0734594a-a8e7-4b1a-a6b1-cd5243059a57' of type Notify

   From what I saw in Wireshark, it seemed that the app would be constantly listening to the ball, and whenever it wanted something done, it would send a message. Therefore, it seemed logical that all commands would be sent to the Write Without Response characteristic and all responses would come from the Notify characteristic. Since there were no other services or characteristics, this configuration also seemed to be the only one possible given the characteristics' types.
4. Because the tutorial only dealt with writing data, I found sample code [10], which showed how to read from a characteristic. I took a piece of data from Wireshark and tried to send it, but received no response. I believe it should be sending okay since there were no errors, but because I don't know how to create a proper packet, I'm not sure.

If the token could be figured out, I think the problem of creating the app would be mostly solved. The other data elements are easy enough to pull from the decompiled code and the Crc32 checksum shouldn't be too hard to generate.

Action Items

To help solve some of the problems outlined in this paper, people need to care more about security and pressure companies to put more effort into securing their products. If customers demand better security, companies will have to implement it if they want to remain competitive and making money. At the moment customers seem to be more focused on cool features and convenience, so there is little pressure to secure devices.

That being said, companies should also take initiative. To be effective, security should be something that gets integrated throughout the development process, rather than something that gets patched as vulnerabilities appear. Companies should have their customers' best interests in mind, which means not exposing them to unnecessary threats. One of the easiest and probably one of the most helpful would be encrypting packets. That way sniffing messages would be much more difficult, if not impossible, without the key. It would also add a layer of difficulty to anyone trying to reverse engineer, and possibly take control of an unsecured device.

Finally, programs that teach computer science should also add security into their classes. We can't expect programmers to be able to implement security if they've never learned it. With a solid background, they'd be more likely to think about potential issues when they code software.

Conclusion

As you can see, Bluetooth devices are vulnerable to attack, and now more than ever their software should be developed with security in mind. Bluetooth 5 was just released, and since it has extra emphasis on integration with IoT devices, those devices will be much more exposed to attack. Reverse engineering can be used to test devices for vulnerabilities by allowing researchers to control inputs directly. Even something as simple as encrypting data being transmitted would significantly reduce the chances of sensitive information being sniffed. I sincerely hope that in the future developers give more thought to the security of Bluetooth devices.

[1] "Bluetooth 5 Now Available." *Bluetooth SIG, Inc.*, www.bluetooth.com/news/pressreleases/2016/12/07/bluetooth-5-now-available, Accessed 12 Dec. 2016

[2] Krebs, Brian. “DDoS on Dyn Impacts Twitter, Spotify, Reddit.” *Krebs on Security*, krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/, Accessed 12 Dec. 2016

[3] Townsend, Kevin. *Microcontrollers GATT Structure*. Adafruit, learn.adafruit.com/assets/13828, Accessed 13 Dec. 2016

[4] Shaked, Uri. “Reverse Engineering a Bluetooth Lightbulb.” *Medium*, medium.com/@urish/reverse-engineering-a-bluetooth-lightbulb-56580fcb7546#.76etuddx6, Accessed 29 Oct. 2016

[5] *APK Downloader*. Evozi, 2016, apps.evozi.com/apk-downloader/. Accessed 2 Dec. 2016

[6] Rukin, Andrew. *Android Apk decompiler*. 2016, www.javadecompilers.com/apk, Accessed 2 Dec. 2016

[7] Shaked, Uri. “Start Building with Web Bluetooth and Progressive Web Apps.” *Medium*, https://medium.com/@urish/start-building-with-web-bluetooth-and-progressive-web-apps-6534835959a6#.k2hs5euye, Accessed 29 Oct. 2016

[8] Shaked, Uri. “Smart Light Bulb Web App (using Web Bluetooth).” github.com/urish/web-lightbulb, Accessed 5 Dec. 2016

[9] “Get Characteristics Sample.” *Web Bluetooth*, googlechrome.github.io/samples/web-bluetooth/get-characteristics.html, Accessed 6 Dec. 2016

[10] “Battery Level Sample.” *Web Bluetooth*, https://googlechrome.github.io/samples/web-bluetooth/battery-level.html, Accessed 6 Dec. 2016