DEVELOPMENT

Bluetooth Low Energy Security

Security... What is it good for...? Absolutely nothing... Or at least, that's the impression I get from the Internet of Things nowadays.

Last night, I was sent this article about security flaws in Samsung's Smart Home which could let strangers unlock your doors. It's a pretty non-trivial hack to figure out, but the risk/reward in being able to get into homes is also very high.

Note: Amusingly, I wrote this post months ago - and just coincidentally, as I was finishing it up yesterday, I received that link and had to re-write my intro...

Anyways, what I want to talk about is BLE security. Specifically, various ways to lock (and not lock) down a BLE-controlled 'device'.

For example, let's imagine I have a BLE-controlled safe or similar (Googling BLE Safe led me to this site: lifestylelock.com). Let's say that there are no physical keys or keypads, it's all BLE-controlled... How can we unlock and get into the safe?

No Security

This method has zero security, and is intended just as a convenience for developers. Create a GATT characteristic described as "Safe Unlocker" which locks when you write a 1, and unlocks when you write a 0.

That's it, you have a BLE-controlled lock! ... An incredibly crappy lock, but a lock nonetheless.

A lot of the uninitiated believe that this is some sort of security, because there is a special GATT characteristic with a 128-bit UUID that needs to have a special number (0, 1) sent into it. The problem is, this is more like security through obscurity - if that. You have a product with 1 GATT characteristic that accepts a number... Not hard to figure it out.

To obscure the usage even more, you could create 20 useless GATT characteristics alongside the 1 correct one, and instead of 0 and 1 to actuate the lock - change that to 871239 and 13123987... I mean, how could anyone guess that?

Well... They probably wouldn't be able to guess it very quickly, but they don't need to. There is one easy method to figure it out, which I'll get to later. Another easy method to avoid the need to guess is that if you have an Android app, that app can be decompiled, and it will show your UUIDs and actuation codes in plain text for the world to see.

An even worse problem with this No Security method is that if you use something like a BLE121LR and you forget to clamp down the transmitting range, someone could lock/unlock your stuff from half a kilometre away!

Custom PIN code

Okay, so if we can't use one generic actuation code, what about letting each user create a custom PIN...? Banks use that for ATM cards, right? Must be SUPER secure!

Well... I won't get into the bank situation, but generally speaking... No, PINs are not secure (for multiple reasons), but they're 'secure enough' for most banks.

Anyways, back to the lock. What if we have a mode on this lock that allows a user to set their own 4-digit pin? That way, we're not worried about someone decompiling an Android app and then being able to open all locks everywhere.

That's much better! Well... No. Still not.

While this custom PIN method at least stops 1 PIN code from being used everywhere, there is absolutely no security surrounding the value of the PIN.

By that, I mean someone with a $30 Bluetooth sniffer could be within range of you, while you're locking or unlocking your safe - and capture your PIN code out of the air. Your safe will still lock and unlock correctly, and you'll be none-the-wiser, but some malicious thief lurking in the bushes will have your safe's PIN code...

In this case, the thief still needs to get into your house, but instead of a safe, let's pretend this is the security you use for your car, or your front door...

> *"Would you send plain-text passwords over HTTP? Then why are you doing it with unpaired BLE? #startup > #security > #consultinglife*
>
> *— Suresh Joshi (@SJoshi84) February 26, 2016"*

Hopefully you're terrified at the thought of that, and are eagerly awaiting a solution.

Well, the solution is very simple.

Pearing... Err... Pairing...

I screw that up every time...

I talked about the pain and suffering that can sometimes be involved in the pairing and unpairing process, but from a security point of view, it's totally worth it. Pairing (and bonding) creates a secure, encrypted link between your phone and your safe - so that when you put in your PIN code, the data is transferred in a way only you and the safe understand.

Those thieving bastards in the bushes will have no idea what hit them.

So, there you have it, all problems solved - life is good, and totally secure... No problem! Everything is perfect.

... Ahem...

Okay, so, maybe not. While we've solved the problem of people sniffing the PIN code out of the air, we haven't solved the underlying problem of using a 4-digit number to unlock a safe.

While your PIN code might be secure when you use it, what's to stop someone from just trying all possible 4-digit permutations and cracking the combo?

Well, nothing... You know how I know? I did it.

I tried it on a custom piece of hardware, but whatever. Didn't take that long either. Even if you assume only 10 attempts per second, you can crack a 4-digit PIN in under 17 minutes. And, since it's BLE, you can do it from far away while it looks like you're just on a phone call.

The attack vector here is that we're protecting our use of the PIN code, however, we haven't stopped anyone from just pairing to the safe and trying all combinations.

Okay, So Now What?

The solution is actually pretty simple - just don't let other people pair. That's it!

Quite often, this solution involves being able to get to the physical device - but let's say there was a button on the inside of the safe that put it into pairing mode for 30 seconds, and I have to one-time pair my phone to the safe in that 30 seconds. This is getting closer to Bluetooth Classic, with the keycodes that show up on each device - to confirm the pairing people know who each other is, out-of-band.

After that timeout, all new pairing attempts are rejected. Your phone has already paired, so it has passed the gatekeeper - and no one else is allowed to pair, and your safe REQUIRES a paired connection, otherwise the BLE stack rejects GATT writes...

You can get rid of the PIN code if you want and go back to the 0 and 1, because the bulk of your security comes from the ABILITY to pair, rather than some hidden information method.

Although, for the sake of marketing, you might want people to have the appearance of safety, letting them type in their own PIN code.

It's Not An Ideal World

In an ideal world, your security comes from the ability to pair. However, this is not an ideal world.

From a usability point-of-view, some people don't like the idea that they NEED to physically do something to explicitly link to a device to use it. Also, they might think this requirement is limiting. For example, let's say I have friends house-sitting, and there is something important I left for them, but I left it for them in the locked safe. They can't get in to push the button to start pairing, so what can they do?

Nothing, except break into the safe. Or... You can re-implement the 'anyone can pair, but they need a PIN code' method.

Stopping the Brute Force Attack

So, I already said that I broke into a device in less than 17 minutes. 17 minutes isn't a big deal to wait. BUT, what if we make it a 6-digit code? Or 8-digits? Or, 8 alphanumerics?

Well, all of those will require more time to break into said device - and here is a website with the numbers. That website assumes 1000 guesses per second - which in BLE-land should be closer to 10-50 guesses per second.

But we're back to usability - how many people want to exchange 8-digit alphanumeric codes? How many people can remember them? Etc... There are lots of ways to solve each of these problems, but they all require more and more work... For instance, 4-digit user entered code + 4 digit random code, all cached on the phone, and an app that will share this info automatically.

That's a decent chunk of work to try to work around a security flaw.

My Solution/Suggestion

Next week, I'll present a suggestion to this BLE brute force problem - as well as some BGScript code that implements it.

Feature Photo credit: Billy Lindblom / Foter / CC BY