CM2303 Essay
Can Inexpensive IoT Devices be Trusted?

David Buchanan, C1673152

November 21, 2017

Contents

1 Introduction

Internet of Things (IoT) devices are becoming increasingly popular[5] in society, especially for domestic use. As these devices become more ubiquitous prices inevitably fall, but at what cost? In this essay, I will be analysing the security of a certain unbranded "WiFi SD Card" product which I purchased over the internet, for relatively low price of £10. I will discuss what issues exist in this product, and how they could be mitigated. Finally, I will discuss security risks of IoT devices in general, and what can be done to improve the situation and protect consumers.

2 "WiFi SD Card" Overview

The device in question is relatively simple. It can be used just like a traditional SD card, for example to store photos from a digital camera. It stores this data to an internal Micro SD card. The "WiFi" aspect of the device refers to the fact that it is capable of creating (or connecting to) a WiFi access point, which in conjunction with a client Android application can be used to remotely access the files.

Figure 1: Android application screenshots.[11]

Before any more advanced analysis, some useful information can be found on the Google Play Store page for the application[11]. A screenshot (Fig. 1) shows a configuration menu where a `guest` and `admin` username and password can be set. This first impression looks positive, however this feature is nowhere to be found in the application itself. The Play Store changelog tells us why:

> **1.Remove login function**
> 2.Fixed Some bugs

Emphasis mine. The current version of the application hard-codes the credentials `admin:admin`, which cannot be changed by the end user. This incident exemplifies a significant portion of the issues facing IoT devices in general - Security is sacrificed to make things simpler/easier for the end user.

The WPA2 WiFi password itself defaults to `99999999`, although fortunately this can be modified by the user in the settings. Statistically however, I would assume that the majority of users simply leave it at the default setting. Alternatively, the user can configure the device to connect to an existing WiFi access point.

3 Protocol Analysis

As mentioned earlier, the device communicates with an Android client application to share the file data. In order to understand how this protocol worked I used both static and dynamic analysis techniques. I started my research by inspecting the Android application.

The majority of Android applications (including this one) are written primarily in the Java programming language. Java is typically compiled to a bytecode representation, which in many cases is trivial to decompile back into the "original" Java code. This process is not perfect, and some semantic information is lost during the compilation/decompilation process, which makes the job of Reverse Engineering a bit harder.

After installing the application[11] on my smartphone, I extracted the "APK" archive which contains all the data used to install the application. I used JADX[9] to decompile the code contained within the APK archive. This process allowed me to confirm that the `admin:admin` credentials were hard-coded into the application. It also gave me valuable information about offsets of various fields within the protocol packets, which I was able to use later to reverse-engineer the details of the protocol. I also discovered the existence of an undocumented "feature", in the form of a partially implemented web server, which is exploitable as I will describe later in this essay.

Deriving all the protocol details from the decompiled application source code alone is certainly possible, but I also wanted to capture some live network packets to aid my analysis. In order to capture packets from my phone, I decided the easiest option was to run the `tcpdump` tool on-device (which requires root privileges). Contrary to what the name might suggest, it is also capable of capturing UDP packets[1]. I used Wireshark[12] to inspect the contents of the UDP packets sent to and from the device. Using a combination of this information, and the information from the decompiled APK, I was able to completely Reverse Engineer the proprietary protocol.

On the topic of traffic inspection, since this protocol is completely unencrypted, anyone who is able to monitor the traffic[10] would be able to view any files transferred to the Android application.

4 Bugs and (In)security

During the process of analysing the protocol, I found numerous issues and vulnerabilities in the product and its companion Android application. The client announces its presence on a network by broadcasting UDP packets to all devices on the LAN. If a malicious user detects the application running on a network, they can deploy a fake SD card server and push arbitrary files to the device (Since the application has no mechanism to authenticate the SD card).

The client application also contains what appears to be an unfinished web server. Attempting a request to this server causes a null pointer dereference inside the application,

[1]The entire protocol was implemented using UDP, presumably because it is easier to implement on very simple hardware (an 8051-based microcontroller)

crashing it. An attacker could simply wait for the application to advertise itself on the network, send a request to the web server, and the client will crash instantly, acting as a Denial of Service attack.

The malicious SD server can also tell the client that authentication was unsuccessful. In this case, the client prompts the user for login details, which are then sent in plaintext to the server. In this way, a malicious server can harvest credentials from the client.

The SD card itself also has vulnerabilities. If we ignore the unchangeable default credentials, we can exploit part of the protocol to remotely "brick" the SD card, making it completely unusable to the standard client application. As part of a configuration packet, there is also a field to set the contents of the broadcast packet that the client uses to identify it. By reconfiguring this value, the client application will never be able to discover the SD card on the network again. This is another form of Denial of Service attack.

This device is almost certainly vulnerable to Key Reinstallation Attacks (KRACK)[10]. These are a recently published set of vulnerabilities in the WPA2 specification (and by extension, WPA2 implementations). Although I have not tested the device itself, I think it is highly likely that it is vulnerable when in client mode. An attacker could potentially use these exploits to decrypt WiFi packets sent to and from the device, and since the protocol itself is unencrypted, they would have full access to any files transferred.

There are also some serious non-security bugs present on the SD card. When the card transmits data to the client over UDP, it does not set the checksum field in the protocol headers. This leads to frequent data corruption even when reading small files. If the password field got corrupted during a password change command for example, then a user could be permanently locked out of the card.

In summary, a malicious user can read all the files from the SD card, semi-permanently "brick" the device, push arbitrary files to the client application, steal passwords, and cause the client to crash repeatedly. I've published[2] proof-of-concept code for some of these attacks[2]. I also suspect there are Remote Code Execution vulnerabilities present on the SD card itself, and this is an area I will continue to research, primarily by reverse-engineering the firmware of the device. Initial firmware analysis reveals that the firmware was last updated on 26/08/2014, which is a very long time in the world of information security, and I think it is highly unlikely that it will ever be updated again.

5 Mitigations

The most secure way to use this device, in its current state, is to only use it in "access point" mode, with a single client device connected to it. The default WiFi password should be changed, and the client application should never be run on any other networks. In this configuration, the device appears to be relatively secure (in my non-expert opinion). However, using the device in this way means that the Android client is unable to connect to the internet at the same time, which severely restricts convenience. The biggest remaining risk is that the client application is left running while the user connects to another network. However, the average user is unlikely to ever receive this advice, and continue to use the device insecurely.

[2]The company behind this product is impossible to contact and are no longer supporting this product, the product is relatively obscure, and exploitation requires physical proximity. Therefore, I decided that full disclosure of these vulnerabilities is most ethical to alert potential users.

The majority of the bugs would need to be solved by completely reimplementing the protocol with robust authentication and encryption. The client needs to authenticate itself to the client, and the client needs to authenticate itself to the server. They should establish an ephemeral per-session key (for forward secrecy), which is used to encrypt subsequent data. Realistically, a small or medium company would not have the resources to design and implement such a protocol securely, so the best solution is to re-use existing protocols. For example HTTPS with client certificates, or maybe SFTP, could be used as an alternative to their "hand-rolled" protocol.

The rest of the bugs present such as the web server in the client application, and the WPA2 KRACK vulnerabilities, require security updates in order to fix them. In an ideal world, the manufacturer should provide timely security updates throughout the lifetime of the product, but in reality this just isn't going to happen. As a compromise, it would be great if they open-sourced all of the software and firmware required, so that the community of users can update things themselves[3]. Alternatively, manufacturers could advertise guarantees on their products, such as "guaranteed security updates for at least 5 years". This particular strategy would be most effective for name-brand manufacturers, which customers trust to actually fulfil their promise.

6 IoT Security in General

The most obvious mitigation of course, is to simply never buy or use these devices. Most customers, if accurately presented with the risks and vulnerabilities involved with this product, would hopefully never buy it. I personally believe that educating end-users about the risks is critical for IoT security in general. Some people are calling for mandatory warnings on IoT device packaging, "like we do for cigarette packets"[6]. I think this is a really great idea, but enforcing it could be difficult. Furthermore, cheap devices imported from overseas are unlikely to comply with such regulations.

A similar idea proposed by Mozilla is an "IoT Trustmark"[1]. The idea is that IoT products should be labelled such that consumers can make an informed decision about what they are buying. The key areas that the report identifies as important are "good data practises" (i.e. How customer data is used), "good security practices", "Openness", "Lifecycle management" (i.e. How long it will continue to receive support), "Establishing that the producing organization is trustworthy". These suggestions are a very good idea, although I still think that getting manufactures to adopt this kind of labelling be almost impossible without some form of legislation. Also, some kind independent authority would need to be appointed in order to ensure accuracy of labelling.

One of the main reasons that IoT security is so important is that we let it control incredibly sensitive areas of our lives, from children's toys to critical healthcare devices. This fact, coupled with their ubiquity and poor track-record for security, poses a massive risk to society in general.

One recent example is the rise of the "Mirai" botnet, which became active around the Summer of 2016. The botnet was comprised of thousands of infected IoT devices, the primary infection vector being insecure default credentials. The infected devices were subsequently used to scan for even more vulnerable devices. The botnet was used

[3]The LineageOS project[8], which supports a wide variety of otherwise unsupported Android devices, is a good example of this kind of approach (Although some components and drivers etc. are not open sourced by vendors).

in a series of DDoS attacks, which caused significant downtime for multiple companies and organisations. The total bandwidth of the attacks was reported to exceed 1Tb/s at times[4], which is the highest ever recorded. At one point, the botnet was used to attack the network infrastructure of the entire country of Liberia, which just goes to show how massive the impact of IoT security issues can be. When there is a potential impact on national infrastructure, the argument for aggressive legislation becomes much stronger.

Over the years, there have been many instances where "smart" children's toys have been hacked[7]. These instances are particularly concerning, as many of these devices have speakers, microphones, or even cameras. If an attacker is able to gain access to one of these devices, the consequences are potentially rather disturbing. If parents were made fully aware of the risks, I highly doubt that they would allow these devices into their homes so willingly.

No matter what legislations and product labellings are put in place, security incidents are still going to happen. In these cases, I believe that companies should be fined harshly, because it's the only way to make them take investment in good security practises seriously. The closest legislation we have currently is the General Data Protection Regulation (GDPR)[3], which will apply in the UK from 25 May 2018. The GDPR primarily focusses on the privacy of users' data, and imposes fines of up to €20 million or 4% of annual global turnover, which ever is greater, in the case that private customer data is mishandled. However, this only covers customer information, and things like IoT botnets would certainly not be covered.

7 Conclusion

In conclusion, the myriad of IoT devices people allow into their homes pose a wide range of risks, and should certainly not be trusted in any way (especially not cheap unbranded devices imported from overseas!). Educating users about these risks is critical, so that they can make informed decisions about whether they want to use a particular product. Manufacturers' security practises need to be regulated much more closely, including harsh fines for infractions. An independent authority that reviews security of devices could be a viable alternative to legislation/regulation, again so that consumers can make informed decisions.

References

[1] Bihr, P. [2017], 'A trustmark for iot: A thingscon report'.
URL: *https://www.mozillapulse.org/entry/436*

[2] Buchanan, D. [2017], 'Evil sd emulator'.
URL: *https://github.com/DavidBuchanan314/wifi-sdcf/tree/master/evil-sd-emulator*

[3] *General Data Protection Regulation* [2017].
URL: *https://ico.org.uk/for-organisations/data-protection-reform/overview-of-the-gdpr/*

[4] Goodin, D. [2016], 'Record-breaking ddos reportedly delivered by 145k hacked cameras'.

URL: *https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/*

[5] Greenough, J. [2016], 'How the 'internet of things' will impact consumers, businesses, and governments in 2016 and beyond'.
URL: *http://uk.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10*

[6] Hunt, T. [2017], 'What would it look like if we put warnings on iot devices like we do cigarette packets?'.
URL: *https://www.troyhunt.com/what-would-it-look-like-if-we-put-warnings-on-iot-devices-like-we-do-cigarette-packets/*

[7] Laughlin, A. [2017], 'Safety alert: see how easy it is for almost anyone to hack your child's connected toys – which? news'.
URL: *https://www.which.co.uk/news/2017/11/safety-alert-see-how-easy-it-is-for-almost-anyone-to-hack-your-childs-connected-toys/*

[8] *LineageOS Android Distribution* [2017].
URL: *https://lineageos.org/*

[9] Skylot [2016], 'skylot/jadx'.
URL: *https://github.com/skylot/jadx*

[10] Vanhoef, M. and Piessens, F. [2017], Key reinstallation attacks: Forcing nonce reuse in wpa2, *in* 'Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)', ACM.

[11] *WiFi@SDCF - Android Apps on Google Play* [n.d.].
URL: *https://play.google.com/store/apps/details?id=com.keytech.wifisd*

[12] *Wireshark* [n.d.].
URL: *https://www.wireshark.org/*