Exploit in Smart Devices: A Case Study

Zian Liu[1], Chao Chen[1], Shigang Liu[1], Dongxi Liu[2], and Yu Wang[3(✉)]

[1] School of Software and Electric Engineering, Swinburne University of Technology, Melbourne, VIC, Australia
929319519qq@gmail.com, {chaochen,shigangliu}@swin.edu.au
[2] Data61, CSIRO, Sydney, Australia
Dongxi.Liu@data61.csiro.au
[3] School of Computer Science, Guangzhou University, Guangzhou, China
yuwang@gzhu.edu.cn

Abstract. With the rapid development of Internet of Things (IoT) and smart devices, an increasing number of home security devices are produced and deployed in our daily life. To improve the awareness of the security flaws of these household smart devices, we perform a demo attack in this paper, which utilizes the vulnerability of a security camera to do the exploit. We set up the malicious Wi-Fi environment and our assuming victim in the experiment uses Samsung GALAXY Note 10.1. We demonstrate how to steal the victim's credential log in information after tricking him into connecting to the malicious Wi-Fi. Our experiment shows that those smart devices lack high-standard security. In our experiment, we show it is trivial and cheap to steal the users credential using a malicious Wi-Fi.

Keywords: Smart things · Cyber attack · Information leak

1 Introduction

Internet of Things (IoT) devices, known as nonstandard computing devices that connect wirelessly to a network and have the ability to send sensor data or surveillance information, are popularly used in various industrial manufacturers because it can bring many advantages to their production process such as workflows optimization, costs-saving etc. [13]. Researchers in the community have shown that people's daily life will continue to be effectively changed with the deployment of SmartThings (e.g., home monitoring sensors, surveillance cameras, and displays, vehicles) [19]. However, previous study has shown that IoT devices are vulnerable to cyber attack, and sometimes can be utilized by hackers to amplify the attack [12]. For example, [5] reported 20 vulnerabilities found in Samsung SmartThings Hub, which works as a center for managing multiple smart devices by means of Zigbee, Z-Wave, Ethernet, or Bluetooth. Exploiting of such smart device hub can enable attackers to do many malicious actions such as unlock smart locks, spy on people through smart cameras, disable motion

W. Meng and S. Furnell (Eds.): SocialSec 2019, CCIS 1095, pp. 152–164, 2019.
https://doi.org/10.1007/978-981-15-0758-8_12

detectors, and even control thermostats. Therefore, it is of great importance to study the security problem of real-world home-based IoT devices.

In this paper, we study one of the in-market household security monitoring devices, which is D-Link Mini HD Wi-Fi Camera. We find that the security camera's working process is vulnerable, which could be exploited to leak users' credential information. The working process is as following. Firstly, users need to connect the security camera. The camera constantly uploads monitoring videos to D-Link's Server, after being connected to Wi-Fi. All the live and history videos are stored on the manufacturer's server. Secondly, users log into the mydlink app to view the monitoring videos. The connections in the whole process are based on TLS Protocol to ensure the data sent through the Internet is encrypted. The data should only be decrypted by the user and the server in the communication.

In order to illustrate the security problem of using D-Link Mini HD Wi-Fi Camera, we demonstrate how to attack the camera. The attacker's goal is to intercept the legitimate user's login information. Then, he can log into this user's account, and view the stored video. The assumption in our demonstration is that the legitimate user has already been tricked into connecting to a malicious Wi-Fi controlled by the attacker, through social-engineering techniques. Once the users log into the mydlink apps, their confidential information such as username and password can be retrieved by the attacker. However, our study discovered that even the attacker retrieved this information, he still cannot log into the legitimate user's account due to extra authentication process implemented by the manufacturer. This extra process contains validating more fields in the login request such as timestamp, signature, and etc. Hence, the extra step for the attacker is to figure out how each of these fields can be forged by reverse-engineering mydlink app. Once the attacker has the ability to change these fields to the correct values, he can forge such log in requests to log into the legitimate users' account. The contributions of this paper are:

- We demonstrate how to forge a TLS certificate that D-Link Mini HD Wi-Fi Camera (model DCS-8000LH) can accept.
- We demonstrate how to forge the login request from mydlink app to bypass the server side authentication process. This is done by reverse-engineering the client side mydlink app.

2 Background

2.1 DNS (Domain Name System)

According to [3], the DNS can resolve a given domain into its corresponding IP address. For example, when a user tries to visit a website www.google.com, before the connection is established, the DNS first translates the domain name into the IP address (e.g., 216.58.196.142).

The DNS working process is described as follows. Once the user in the network has a DNS request, firstly the request is sent to the gateway. If the gateway has the cached DNS record, it will respond to the DNS query directly. If the

gateway does not has the cached DNS record, the gateway will do the multiple layered DNS queries as shown in Fig. 1. This process starts from querying the root name server, then layer by layer, to obtain the IP address from the target name server (UWCS in Fig. 1).

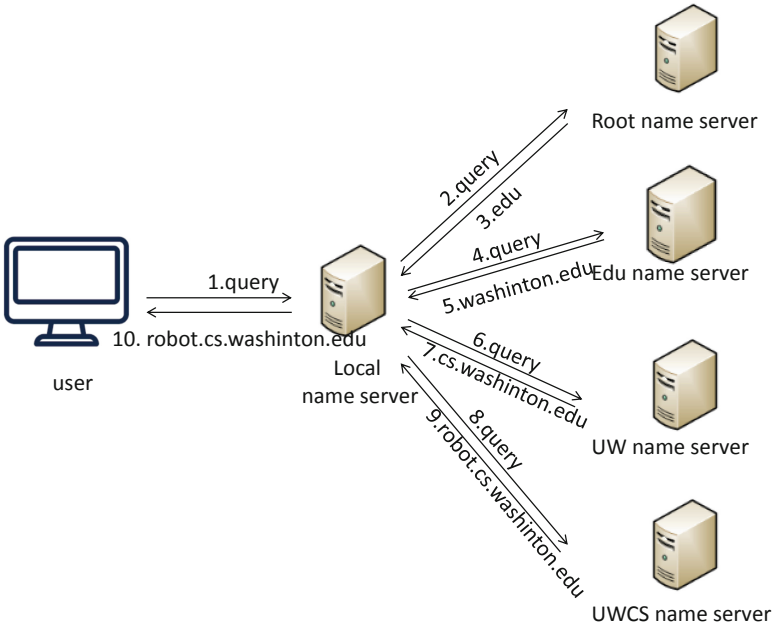

Fig. 1. Multiple layered DNS query

2.2 Transport Layer Security (TLS) Protocol

According to [11], the TLS protocol is a cryptography protocol aimed at improving the computer network's communication security. The server and the client in the TLS protocol use the symmetric key for transmitting data to each other. The symmetric key is negotiated by the client and the server based on the TLS handshake protocol.

The flowchart of the TLS handshake protocol is shown in Fig. 2 [11]. In brief, as in Fig. 2, the steps before the Application Data are referred to as the TLS handshake process. The aim is to make the client and the server agree upon a master secret key, which is used later for generating the symmetric encryption key for the Application Data.

In this paragraph, we briefly introduce the TLS handshake process. As per [11], *ClientHello* and *ServerHello* each contains a random number, which is later used for generating the master secret key and the symmetric encryption key. A

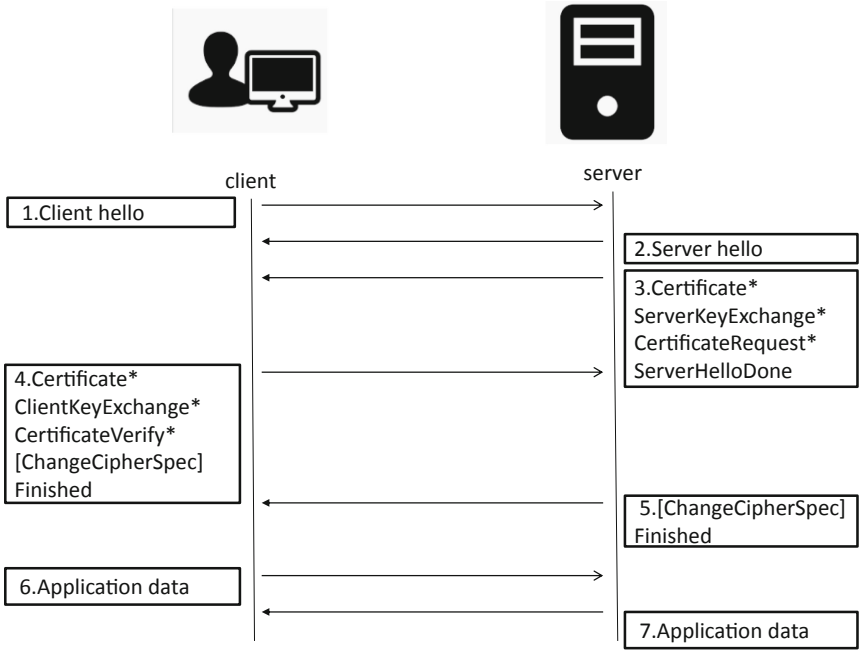

Fig. 2. The TLS handshake protocol

certificate is used for authentication and contains the public key, which can be in either RSA or Diffie-Hellman format. When the public key information is not enough in *Certificate*, *ServerKeyExchange* can provide extra information. After the *ClientKeyExchange*, the client and the server should be able to agree upon a pre-master secret key. Then both sides use this pre-master secret key with the client and server random numbers to generate the master secret key.

2.3 Certificate

As per [15], a certificate is used in the TLS connection for authentication. It binds the identity of a party to a pair of keys. That is, each party can own a certificate, with its public key known to the public, while keeping its private key secret. A party's private key can be used for signing other parties' certificates. Hence, it can be easy to verify that other parties' certificates are signed by that party using the signing party's public key.

2.4 Certificate Authority (CA)

A CA is a party that issues certificates. Each CA owns a certificate that signed by its own private key. This certificate is known as the Root Certificate. This Root Certificate also contains the public key corresponding to its own private

key. The CA's Root Certificate is trusted and installed in most browsers and mobile devices by default. When the browsers or the mobile devices connect to a server through TLS protocol, the server usually provides a certificate. To validate the server's certificate, the public key in the Root Certificate is used to check the server's certificate is indeed signed by that Root Certificate.

2.5 Certificate Chain

A certificate chain represents a chain of trust. There are two categories of CAs: root CAs and intermediate CAs. To validate a TLS certificate chain, each node in the chain must be signed by its preceding certificate, and the first node should be signed by Root Certificate. For example, as depicted in Fig. 3 [4], suppose that we have installed the Root Certificates in the browser or the mobile device. We can use the Root Certificate's public key to validate that the Intermediate Certificate is signed by the Root Certificate. Then we can use the Intermediate Certificate's public key to validate that the End-entity Certificate is signed by the Intermediate Certificate. In a similar way, we can validate the remaining part of the certificate chain block by block.

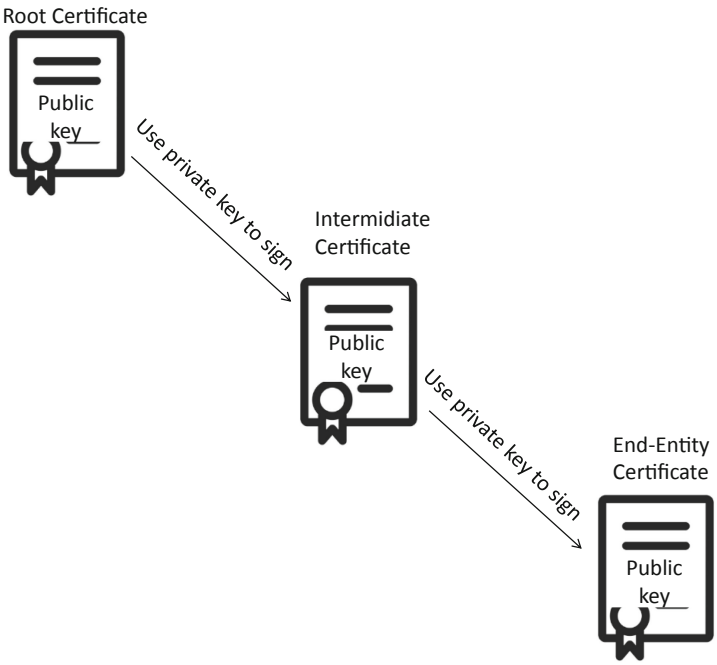

Fig. 3. The trust chain

2.6 Hypertext Transfer Protocol Secure (HTTPS) Protocol

From [16], HTTPS protocol is encrypting the HTTP data by TLS or Secure Sockets Layer (SSL). So HTTPS is also referred to HTTP over TLS or HTTP over SSL. Due to the encryption of the TLS or SSL, HTTPS is considered secure against the man-in-the-middle attack. Analog to HTTP protocol, HTTPS also provides the same set of requests for the client (e.g., GET, POST, PUT, and etc.).

3 Related Work

Many works have been done concerning the cyber attacks on IoT devices. Some of these works focus on the vulnerabilities within the devices, others need the participation of a controlled Wi-Fi. [18] introduced the risk of connecting to a public Wi-Fi, including the attack of Sniffers, Evil Twin, Man-in-the-Middle Attacks, and Sidejacking.

Jyoti et al. [10] summarised the general attack approaches on the IoT devices. This includes physical attacks (e.g., node tampering, RF interference on RFIDs, node jamming in WSNs, malicious node injection, physical damage, social engineering, sleep deprivation attack, and malicious code injection), network attacks (e.g., traffic analysis attacks, RFID spoofing, RFID cloning, RFID unauthorized access, sinkhole attack, man in the middle attacks, denial of service, routing information attacks, sybil attack), software attacks (e.g., phishing attack, virus, worms, trojan horse, spyware and adware, malicious scripts, denial of service), encryption attacks (e.g., Side-Channel attacks, Cryptanalysis Attacks).

Gunes et al. [6] presented two web-based attacks against IoT devices on the local area network (LAN). In the scenario, since the victim IoT devices have open HTTP servers, once they visit a website containing malicious codes, they can be infected.

Zoran et al. [8] introduced the essence of Main In the Middle (MITM) attack used in IoT devices. In this paper, they also included common methods used in a MITM attack. Mauro et al. [9] did a survey on the MITM attack based on different spoofing techniques and different network protocols.

[7] described the Domain name Server (DNS) service and its relevant vulnerabilities, which include cache poisoning. Simar et al. [17] demonstrated various DNS spoofing approaches in more detail in their paper.

Even though the TLS protocol is proved to have high security, it still contains several vulnerabilities. Chris et al. [14] mentioned some of these vulnerabilities, for example, lack of Certificate Pinning.

4 Attacking Demo

4.1 Environment Setting

Gateway. Our gateway is a RaspberryPi, which is connected to the desktop through the SSH protocol, in order to access the Internet. This RaspberryPi also

acts as a hotspot, which creates a Wi-Fi, by running the open-sourced program create_ap [1]. In the experiment, our gateway IP address is 192.168.12.1.

DNS Caching. To perform the DNS spoofing, the attacker needs to install the malicious DNS cache into the gateway in advance. To achieve this, when running the create_ap program, we set this program to take into account additional hosts file using the command "-e". Hence we created this additional host file named dnsmasq.hosts, which contains 4 lines:

192.168.12.1	mp-sg-dcdda.auto.mydlink.com
192.168.12.1	api.auto.mydlink.com
192.168.12.1	mp-sg-openapi.auto.mydlink.com
192.168.12.1	mp-sg-dcdda.auto.mydlink

As shown in these lines, when answering the DNS requests, the gateway will resolve the domain names in the right column into the IP address in the left column. Therefore, all these domain names will be resolved into the IP address, 192.168.12.1, which is our gateway.

To open up a hotspot named liuzian with DNS spoofing capability, the attacker types the following line into the command window in the RaspberryPi: sudo create_ap wlan0 eth0 -e /etc/dnsmasq.hosts liuzian.

4.2 Mydlink App

Mydlink is a mobile app designed for managing home security devices, by the smart camera's manufacturer. A user can combine his account with multiple home devices. As long as their home security devices are connected to the Internet, these devices keep sending monitoring data to the server. When users want to view the device's content, they can log into their accounts to manage their devices. The process behind logging into this app is introduced below.

Once a user opens up his mydlink app on the mobile device and tries to log into the account, the app firstly sends the DNS query about api.auto.mydlink.com to the gateway. The gateway will respond to this query by firstly looking up its cached record. Once such a record exists, the gateway will respond using this record. Otherwise, the gateway starts the multilayered DNS query as shown in Fig. 1.

Once retrieved the IP address in the response, the app tries to establish an HTTPS connection to the server with that IP address. To establish the HTTPS connection, the first step is to establish a TLS connection. After the TLS connection is established, the app will send an HTTP request encrypted by this TLS protocol to the server. The request includes the user-typed username and the MD5-encrypted password, along with other tokens such as the timestamp, the signature and etc., for server-side authentication.

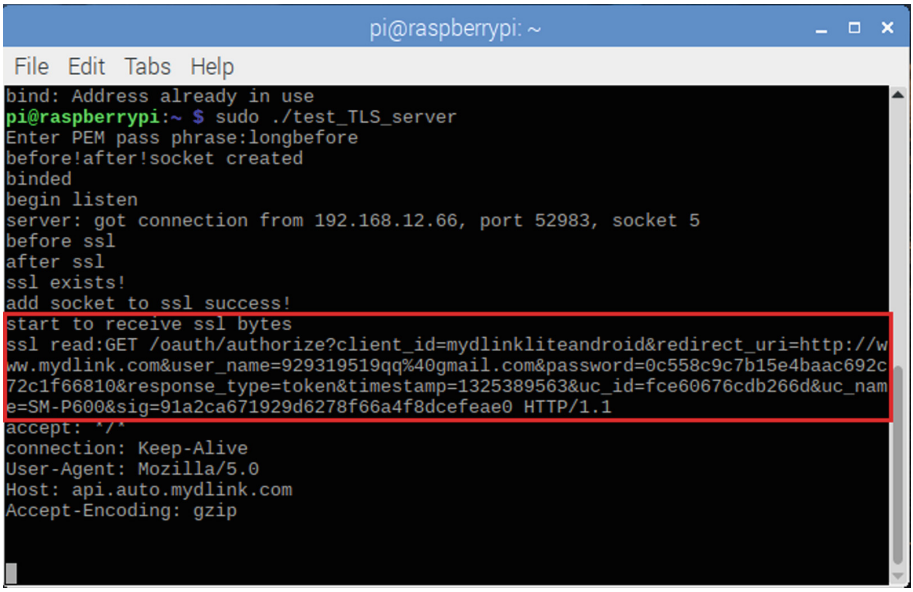

Fig. 4. The account information interception (Color figure online)

4.3 Mydlink App Log in HTTPS Request

As we introduced in the last paragraph, the HTTP request contains many fields. In this paragraph, we will introduce these fields in detail. Figure 4 shows an example of such HTTP request. The HTTP request contains both encrypted fields (e.g., field password is encrypted by MD5 hash) and plain-text fields (e.g., field username). Even though the attacker can obtain the password in cipher form, he is still unable to log into the legitimate user's account, as the server side requires other fields in the request for validation. The HTTP GET request is highlighted in Fig. 4 is as follows:

GET /oauth/authorize?client_id=mydlinkliteandroid&redirect_uri=http://www.mydlink.com&user_name=929319519qq%40gmail.com&password=dd71a460c42b71c6e9f3fdc578bd6c28&response_type=token&timestamp=1547686112&uc_id=fce60676cdb266d&uc_name=SM-P600&sig=ec383858369c41f7b901f41d9765f703 HTTP/1.1

This request contains multiple fields. Among these fields, *client_id*, *response_type*, *uc_id*, and *uc_name* are fixed values depending on the mobile device. They represent the Mydlink app version, response type, device identification, and device name respectively. *User_name* is the plain-text username that the user entered.

Password is the MD5 encrypted account password. *Timestamp* is the integer form of the system time divided by 1000. *Sig* is the MD5 encrypted result of all the string before the tag "&sig", appended with a hard-coded string "83EB5870943E44A1B483C8E9BE5A36BC". The above introductions are all based on the reverse-engineer analysis of the Android mydlink app, which is performed using Android Studio.

4.4 Charles Proxy

According to [2], Charles is an HTTP proxy. It helps the user to debug the HTTP/HTTPS connections by deploying between the user and the server. Installing the Charles Root Certificate on the user side ensures the connection between the user and Charles can be set up. To establish the connection to the server, Charles forges the Certificate Chain according to the original Certificate Chain provided by the user, which maximizes the possibility that the server can accept this certificate.

Once Charles is deployed between the user and the server, it provides multiple functionalities to the user such as blocking requests, modifying requests in both directions and etc.

4.5 Scenario

Our experiment scenario is that the legitimate user is tricked into connecting to a malicious Wi-Fi. Initially, the legitimate user is tricked into connecting to the server in step 1. We deliberately set the fake DNS records as in subsection *DNS caching*. As a result, in step 2, a forged DNS record is used as the response to the DNS query. So the app will regard the gateway as the server and tries to set up the TLS connection with the gateway as in step 3. In order to accept this TLS connection, the attacker runs a TLS server program on the gateway. Due to the app side authentication request, the attacker needs to prepare a forged certificate chain on the server side as 4. After the app side's validation, the TLS connection is established. Now the server side can decrypt the HTTPS data sent in 5 using the symmetric TLS key. The program on the server side is also programmed to be able to record the plain-text HTTPS data through this TLS connection (Fig. 5).

4.6 Attacking Steps

Forging the Certificate. As denoted in Fig. 2, in some cases, within the TLS handshake protocol, the certificate authentication only occurs at one side. Since in our assumed attacking scenario, the attacker uses the fake server (gateway) to send a fake certificate chain to the user, the attacker needs to imitate the original Certificate Chain to forge the fake one. Figure 6 depicts the original certificate chain sent from the real server that is to be validated by the mydlink app in the normal scenario.

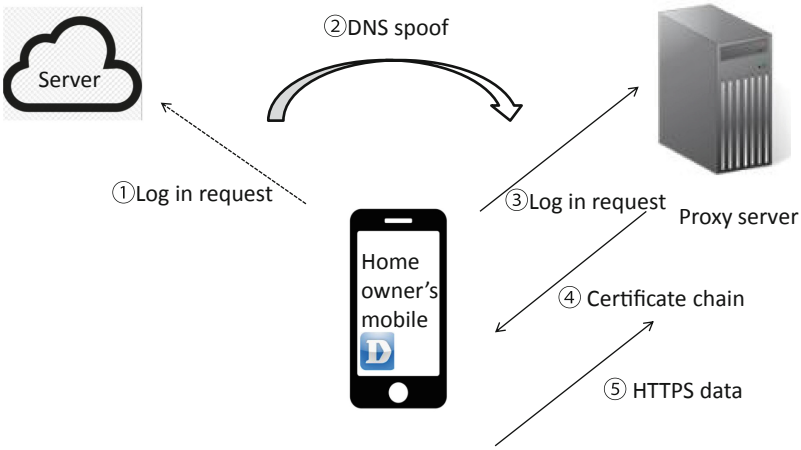

Fig. 5. The flowchart of our experiment scenario

```
Certificate:(id-at-commonName=*.auto.mydlink.com,id-at-organizationName
Certificate
Certificate:(id-at-commonName=GlobalSign Organization Validation CA - SI
Certificate
Certificate:(id-at-commonName=GlobalSign Root CA,id-at-organizationalUn
```

Fig. 6. Server's certificate chain in normal scenario

```
(id-at-countryName=BE)
(id-at-organizationName=GlobalSign nv-sa)
(id-at-organizationalUnitName=Root CA)
(id-at-commonName=GlobalSign Root CA)
```

Fig. 7. The Root Certificate

```
(id-at-countryName=BE)
(id-at-organizationName=GlobalSign nv-sa)
(id-at-commonName=GlobalSign Organization Validation CA - SHA256)
```

Fig. 8. The Intermediate Certificate

The server's certificate chain shown in Fig. 6 contains 3 certificates. The Root Certificate is GlobalSign Root, its information is shown in Fig. 7.

The second certificate is signed by the Root Certificate, also known as the Intermediate Certificate. Its information is shown in Fig. 8.

The last certificate in the chain is signed by the Intermediate Certificate. This certificate is owned by the mydlink corporation and also known as the End-entity Certificate. Its information is depicted in Fig. 9.

```
(id-at-countryName=TW)
(id-at-stateOrProvinceName=Taipei)
(id-at-localityName=Taipei)
(id-at-organizationalUnitName=IT Dept)
(id-at-organizationName=D-Link Corporation)
(id-at-commonName=*.auto.mydlink.com)
```

Fig. 9. The End-entity Certificate

Based on the observations described above, we can forge the certificate chain. The forged End-entity Certificate should be forged to at least contain the same value of commonName, organizationName, organizationUnitName, localityName, stateOrProvinceName, and contryName in the legitimate End-entity Certificate. At this stage, we do not know the server side certificate authentication process, which means there might be additional validation on the Intermediate Certificate or Root Certificate. Hence we try to guess how to forge it from the minimum requirements. This minimum requirement assumes that the server side only validates the End-entity certificate's fields, such as commonName, organizationName, organizationUnitName, localityName, stateOrProvinceName, and contryName.

To test and improve our guessing benchmark, we have performed the experiment on an Android device, Samsung Galaxy Note 10.1. Since the Android mydlink app only trusts the system certificate, in the real attack, we can purchase a legitimate Intermediate Certificate from one of the Android trusted root CAs. Then we can use our purchased Intermediate Certificate to sign the End-entity Certificate. However, in the experiment, for simplicity, we replaced this step with an equivalent step, by creating our own Root Certificate and embedding it into the system trusted store. Then we used our created Root Certificate to forge the certificate chain. Our forged certificate chain is depicted in Fig. 10.

```
▷ Certificate: 3082045b30820243a003020102020900 9060fd325e29b0e2... (id-at-commonName=*.auto.mydlink.com,id-at-organizati
  Certificate Length: 1295
▷ Certificate: 3082050b308202f3a003020102021100abc49193d9427913... (id-at-organizationName=Example,id-at-countryName=GB)
```

Fig. 10. The forged Certificate Chain

The forged certificate chain contains 2 certificates. We discarded the Intermediate Certificate in the forged chain because we started our guessing from the minimum requirement, that the server side only validates the End-entity certificate. The Root Certificate's information is shown in Fig. 11.

The End-entity Certificate is directly signed by our Root Certificate. Its information is depicted in Fig. 12.

Intercepting HTTPS Information on the Gateway. As introduced before, since we have forged the DNS cached record, when the Android app tries to set

```
(id-at-countryName=GB)
(id-at-organizationName=Example)
```

Fig. 11. The forged Root Certificate

```
(id-at-countryName=TW)
(id-at-stateOrProvinceName=Taipei)
(id-at-localityName=Taipei)
(id-at-organizationName=D-Link Corporation)
(id-at-organizationalUnitName=IT Dept)
(id-at-commonName=*.auto.mydlink.com)
```

Fig. 12. The forged End-entity Certificate

up the TLS connection and send the user account information to the server, it actually connects to the gateway. On the fake server (gateway), we opened up a C program that will listen to and accept the TLS connecting request. The functionality of this C program is introduced here. After the mydlink app connects to this program, this C program will record all the information sent through this connection in plain-text form. In order to set up the connection with the app, this program will send the forged certificate chain. In our experiment the TLS handshake was successful. The intercepted HTTPS request is introduced in *Mydlink app log in HTTPS request*.

Logging into Legitimate User's Account. At this step, the attacker has obtained the legitimate user's account information, but it is not enough for intruding into the user's account. After reverse engineering, we found that except username and password, the server side also validates the signature and timestamp. Hence to forge the HTTPS request, two more fields (timestamp and sig) need to be modified. Therefore we have implemented a simple Javascript program to refresh the real-time timestamp value for our reference. The timestamp value is updated on the web page every second.

To forge the request, the attacker only needs to calculate the *timestamp* and *sig* in advance. The rules for modifying these two fields are also manually written into Charles by us ahead. For example, since at this stage, we know how the app produces each of these fields if a user sends the login request 5 min later. We can instantly calculate the value of each these fields if it is being sent 5 min later. Then we tell Charles what value each field is to be modified to. Then we manually send the login request in 5 min. Charles will stop the request and modify each field to the desired value before sending it to the server.

5 Conclusion

In this paper, we have demonstrated a possible attack on the Wi-Fi camera. Our assuming scenario is that the attacker can trick the user into connecting to

a malicious Wi-Fi. Then the attacker uses reverse engineering results and the intercepted account data to intrude into the user's account. We have shown that even though the communication protocol the smart device based on is secure enough, a cyber attack is still possible using other measures together with social engineering techniques. The smart device manufacturers should implement a more complex authentication process for the TLS certificate on the app side. The users should always be aware of the Wi-Fi security they are connected to.

References

1. https://github.com/oblique/create_ap . Accessed 05 Dec 2018
2. https://www.charlesproxy.com/. Accessed 24 Nov 2018
3. What is DNS?—How DNS works. https://www.cloudflare.com/learning/dns/what-is-dns/. Accessed 20 Nov 2018
4. How certificate chains work (2018). https://knowledge.digicert.com/solution/SO16297.html. Accessed 28 Nov 2018
5. Researchers reveal 20 vulnerabilities in Samsung Smartthings Hub (2018). https://www.csoonline.com/article/3292942/researchers-reveal-20-vulnerabilities-in-samsung-smartthings-hub.html. Accessed 28 Jan 2019
6. Acar, G., Huang, D.Y., Li, F., Narayanan, A., Feamster, N.: Web-based attacks to discover and control local IoT devices. In: IoT S&P@SIGCOMM (2018)
7. Al-Hajeri, A.: DNS spoofing attack support of the cyber defense initiative (2014)
8. Cekerevac, Z., Dvorak, Z., Prigoda, L., Cekerevac, P.: Internet of things and the man-in-the-middle attacks-security and economic risks. MEST J. **5**(2), 15–25 (2017)
9. Conti, M., Dragoni, N., Lesyk, V.: A survey of man in the middle attacks. IEEE Commun. Surv. Tutor. **18**(3), 2027–2051 (2016)
10. Deogirikar, J., Vidhate, A.: Security attacks in IoT: a survey. In: 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), pp. 32–37, February 2017
11. Dierks, T., Allen, C.: The TLS protocol version 1.0. Technical report (1998)
12. Kolias, C., Kambourakis, G., Stavrou, A., Voas, J.: DDoS in the IoT: Mirai and other botnets. Computer **50**, 80–84 (2017)
13. Lee, I., Lee, K.: The Internet of Things (IoT): applications, investments, and challenges for enterprises. Bus. Horiz. **58**, 431–440 (2015)
14. Stone, C.M., Chothia, T., Garcia, F.: Spinner: semi-automatic detection of pinning without hostname verification, pp. 176–188 (2017)
15. Prodromou, A.: TLS/SSL Explained – TLS/SSL Certificates, Part 4 (2017). https://www.acunetix.com/blog/articles/tls-ssl-certificates-part-4/. Accessed 23 Nov 2018
16. Rescorla, E.: HTTP over TLS. Technical report (2000)
17. Preet Singh, S., Maini, A.: Spoofing attacks of domain name system internet (2011)
18. Private WiFi: The hidden dangers of public WiFi (2014)
19. Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M.: Internet of things for smart cities. IEEE Internet Things J. **1**(1), 22–32 (2014)