Penetration Testing: IoT Light Bulbs

ECE419/CS460 Final Project
Spring 2018

Bill Xun, Jonathan Hsun

1 Introduction

The Internet of Things (IoT) is a growing technological phenomenon that has the potential to greatly advance and improve the lives of all people. The Internet of Things brings regular objects to life with embedded electronics, and are then controlled via Bluetooth or Wi-Fi remotely. However, more often than not, companies rush to meet deadlines and push products without addressing the security aspect of their products. With the increasing connectivity of some devices, connected services may not always be secure. As seen by the countless IoT hacks in recent years, and with 200 billion connected sensor devices are currently in use in the world today[1], more must be done to secure these devices.

The goal of our project is to investigate the possible security vulnerabilities in IoT connected light bulbs, and see if they can be exploited easily at home, without sophisticated hardware or reversing, or black-box testing, like using a drone to exploit Hue lights. Our main motivation for the project stems from our own exposure to these IoT devices. We both own Philips Hue bulbs, so we decided to examine the vulnerability of the device.

Attempting to find exploits of our own, we soon realized that the bulb has been patched and updated extensively and most if not all simple vulnerabilities have been resolved. As a result, we decided to investigate security vulnerabilities in a cheaper, yet popular, branded bulb, the MagicLight Bulb. Produced by a much smaller company than the Philips Hue, the MagicLight bulbs provided us with a rather insecure platform that many other IoT devices from small companies might resemble. We were able to find several issues which allowed almost total control of a bulb inside and outside of the network its connected to. The results of our findings are detailed in this paper.

[1] Scott R Peppet, "Regulating the Internet of Things: First Steps Toward Managing Discrimination, Privacy, Security, and Consent." *Texas Law Review* 93 no. 1 (2014): 85-178.

2 Philips Hue

2.1 Vulnerability analysis

As per our original project proposal, we began by attempting to find a vulnerability in the Philips Hue lightbulb. The Hue, like any other IoT device, is connected to the internet through multiple services shown in Figure 1. These services offer possible attack vectors that may be explored. That being said, the services and their communication with Philips Hue bulbs are adequately secure. More specifically, communication between these services and the bridge are secured using TLSv1.2 encryption.

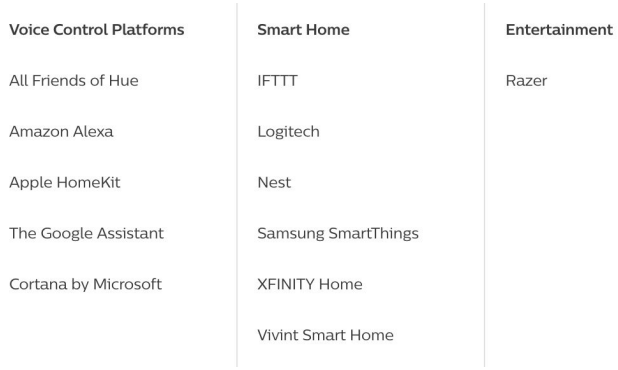

Figure 1: Connectivity of Philips Hue

From here, we attempted to look at two lines of traffic communication:

1. Bridge to bulb. The bulbs connect to the bridge by a protocol called Zigbee that uses the 802.15.4 protocol. As a result, we are unable to sniff packets without the right hardware (which can be conveniently purchased for as little as $500).
2. Router to bridge. We managed to capture packets sent between the bridge and a router because communication is done in HTTP. At first glance, the traffic may seem insecure; however, the payload of each packet is encrypted with AES which we verified through some research.

2.2 Attempts at Exploits

After assessing the vulnerabilities of the Hue, we looked into the Hue developer tools provided by Philips. Requiring physical access to the hue bridge itself, the developer API can be accessed by following steps[2] listed on the Hue developer website. The interface itself is loaded at http://<bridge ip address>/debug/clip.html and http requests may be sent directly to the bridge. While powerful, this tool can only be accessed by following steps that require access to the physical 'link' button on the bridge. The bulb also has a web interface at https://my.meethue.com/en-us/my-devices that

[2] https://www.developers.meethue.com/documentation/getting-started

allows a user to turn on and off the lights. Fuzzing the website yielded nothing of interest. Likewise, we attempted to fuzz the web server, that also yielded nothing interesting.

We also found a published exploit on an old version of the Philips Hue firmware[3]. Essentially, a lack of encryption in traffic communicating with the public API allows attackers to read API keys by sniffing HTTP traffic. This, among other known exploits, have since been patched by philips in subsequent firmware updates.

2.3 Rooting the bridge

After some research, we learned about the possibility of rooting the Hue Bridge[4]. In order to do so, one must have physical access to the bridge, after which a serial to USB cable can be connected to pins on the internal chip (Figure 2).

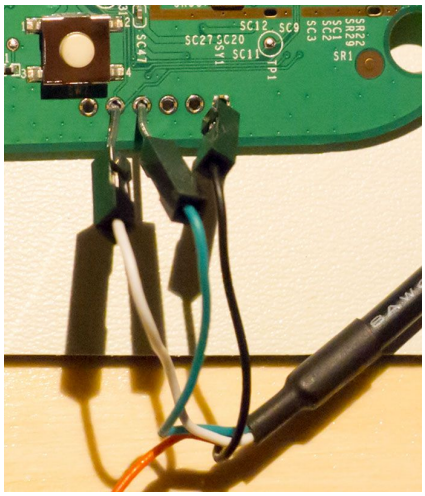

Figure 2: Hue bridge pins

If done correctly, the Hue bridge will boot into debug mode and can be controlled by a connected computer. From here, it is possible to set the bridge's iptables and open up the telnet port. In addition, one can add their public ssh key as an authorized key and retain remote access to the console.

2.4 Final notes

The Hue bulb assumes full trust in local devices and open APIs. While this allows the bulb to be programmed easily through tools on trusted devices, it also opens up a whole new vector of attacks not targeted at the system itself, but rather the devices it trusts. Personal computers on the same network as the hue may be compromised instead, and the hue is not able to differentiate a malicious command from an intended one when coming from a device it assumes is secure. Our project focuses on the exploitability of the actual IoT device, and although there may exist possible exploits requiring access to a Hue trusted device, we wanted to focus on the IoT device itself.

[3] https://www.cvedetails.com/cve/CVE-2017-14797/

[4] http://colinoflynn.com/2016/07/getting-root-on-philips-hue-bridge-2-0/

Exploiting the philips hue was mainly explored in the time period before our project status report, and after realizing the Hue was sufficiently secure (at least enough to deter amateur hackers like us), we moved on to the MagicLight bulb.

3 MagicLight

3.1 MagicLight

The Magic Bulb purchased from Amazon is a wifi-enabled RGB light bulb capable of being remotely controlled from anywhere with internet. Unlike the Hue, this bulb has open ports directly connected to the internet, making it much more vulnerable to exploits. The bulb, unlike the hue, does not require a bridge or intermediate device to operate as it contains a ESP8266 Wi-Fi module allowing it to connect directly to the internet.

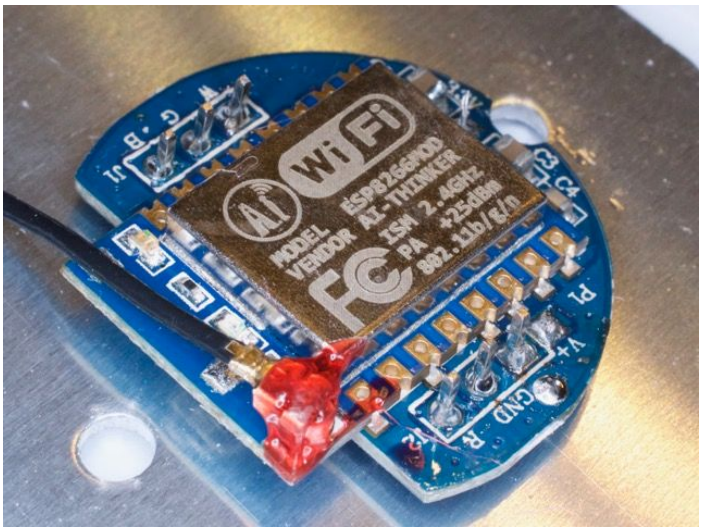

Figure 3: The ESP8266-07 Wi-Fi module[5]

We investigated how the bulb communicates with the app from within the network and remotely, as well as the process in which it is setup and connected to the internet, and registered to a user. The results of our analysis are below.

[5] https://www.espressif.com/en/products/hardware/esp8266ex/overview

3.2 Local Communication

Using wireshark, we investigated how the bulb communicates when running on the same network as the app. The bulb has a wifi-adapter and some sort of a network server running on port 5577, found using nmap. The app initiates a TCP connection to the bulb which is assigned a (static or dynamic) IP address on the router. The app sends hex bytes to the bulb, which contain commands to the bulb.

1807	42.484959	10.10.123.5	10.10.123.3	TCP	54 56511 → 5577 [ACK] Seq=1 Ack=1 Win=65535 Len=0
1808	42.491311	10.10.123.5	10.10.123.3	TCP	58 56511 → 5577 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=4
1809	42.493711	10.10.123.3	10.10.123.5	TCP	68 5577 → 56511 [PSH, ACK] Seq=1 Ack=5 Win=5836 Len=14
1810	42.493764	10.10.123.5	10.10.123.3	TCP	54 56511 → 5577 [ACK] Seq=5 Ack=15 Win=65535 Len=0
1811	42.497183	10.10.123.5	10.10.123.3	TCP	66 56511 → 5577 [PSH, ACK] Seq=5 Ack=15 Win=65535 Len=12
1814	42.579741	10.10.123.3	10.10.123.5	TCP	54 5577 → 56511 [ACK] Seq=15 Ack=17 Win=5824 Len=0
2136	48.963122	10.10.123.5	10.10.123.3	TCP	62 56511 → 5577 [PSH, ACK] Seq=17 Ack=15 Win=65535 Len=8
2160	49.187174	10.10.123.3	10.10.123.5	TCP	55 5577 → 56511 [PSH, ACK] Seq=15 Ack=25 Win=5816 Len=1
2161	49.187289	10.10.123.5	10.10.123.3	TCP	62 56511 → 5577 [PSH, ACK] Seq=25 Ack=16 Win=65535 Len=8
2162	49.212108	10.10.123.3	10.10.123.5	TCP	55 5577 → 56511 [PSH, ACK] Seq=16 Ack=33 Win=5808 Len=1
2163	49.212202	10.10.123.5	10.10.123.3	TCP	54 56511 → 5577 [ACK] Seq=33 Ack=17 Win=65535 Len=0

Fig 4: Screenshot of communication between the app and the bulb over TCP on port 5577 on Wireshark

The app sends hexadecimal code commands to the bulb, to which it responds with 0x30, which is an ack to our understanding.

```
00000010  31 ff bf 58 00 f0 0f 46
0000000E  30
00000018  31 ff ff ff 00 f0 0f 2d
0000000F  30
```

Fig 5: A hexdump of the TCP packets between the app (red) and the bulb (blue)

3.3 Hex Code Analysis

To figure out how the hex commands to the bulb work, we sent various commands to the bulb and compared the hex codes. In particular, we sent commands for white, red, green, blue, various colors on the RGB wheel, dimmed colors, and commands to turn the bulb on and off.

Eventually we saw similarities and came up with the following mapping to the hex codes:

Change Color: 31 [R] [G] [B] 00 f0 0f [2 byte checksum] *where R,G,B are 0-255 in hex*
Turn the bulb ON: 71 23 0f a3 -- *where a3 is the 2 byte checksum*
Turn the bulb OFF: 71 24 0f a4 -- *where a4 is the 2 byte checksum*

31 appears to be the OP code to change color, and 71 the op code to turn the bulb on and off.

We suspected the last 2 bytes were some sort of checksum, because after figuring out the R,G,B values, the commands would not go through unless the last 2 bytes were correct. Running the hex commands without the last 2 bytes through a list of checksums online[6], we discovered that the checksum used was: *(sum of all the bytes in hex) % 256.*

3.4 Remote Connection

We also investigated how the bulb is remotely controlled. Using wireshark again, we discovered that the commands are sent to a remote server, 47.88.101.237.

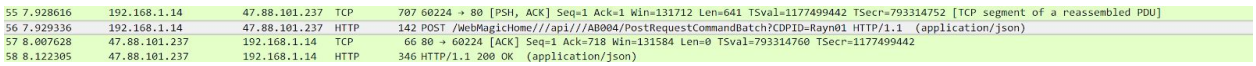

Fig 6: HTTP requests from the app to the remote server

The app makes a POST request to an HTTP server that runs the ASP.NET framework on port 80. The web server runs on HTTP instead of HTTPS, meaning communication to and from the server is unencrypted, and can be seen by everyone. Consequently, we were able to clearly see how requests to control the bulb are made.

```
POST /WebMagicHome///api///AB004/PostRequestCommandBatch?CDPID=Rayn01 HTTP/1.1
Host: ryan.magichue.net
Content-Type: application/json
Cookie:
.ASPXAUTH=8B8CEDE871EE5E49FC85CA12EE9B9AB0E06CFD7D6A695DDB10794BDA590B783AE17C7725D
3DDD971D17A4ABD8D2223401AEE96FCB6FB99B9D84E3090ED4B9091E3DE8E652D2F2479000C61100450
8312A32197CE45466507559EA44B6C8BF72A0480EE77759A0CC2C28C908235FE7C528DE5C1D1BFFBABA
2A643DC14C1D599DD
Connection: keep-alive
Accept: */*
User-Agent: LedWiFiProRyan/1.0.9 (iPhone; iOS 11.3; Scale/3.00)
Accept-Language: en-GB;q=1, en-SG;q=0.9, zh-Hans-SG;q=0.8, ko-SG;q=0.7
Content-Length: 76
Accept-Encoding: gzip, deflate

{"DevicesCMDs":[{"HexData":"31ff141400f00f57","MacAddress":"600194912715"}]}HTTP/
1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 04 May 2018 03:47:35 GMT
Content-Length: 49
```

Fig 7: The POST request to the server and the server response

[6] https://www.scadacore.com/tools/programming-calculators/online-checksum-calculator/

In particular, the app sends a POST request to ryan.magichue.net (47.88.101.237) with a .ASPXAUTH cookie as authentication, and then sends a series of commands for lightbulbs in json format, with the hex data, and the MAC address of the bulb in ascii, to which the server responds with a 200 OK, and sends the appropriate command to the bulb.

Further analysis into the domain (47.88.101.237) shows that it is an ip owned by HiChina Zhicheng Technology Ltd, located in Hong Kong, on the Alibaba network. A port scan reveals 7 ports open: 80 and 443 corresponding to HTTP and HTTPS, 21 which is a ftp server running FileZilla, and 49165,49154 which seems to be used by Apple's XSAN clustered filesystem, but these ports fall under dynamic/private use.

Ports 8805 and 8806 seem to be ports where the bulbs communicate with the server, as this port is sent to the bulb from the app during initialization (refer to Section 3.5 below).

Additionally, if the .ASPXAUTH cookie is obtained, sending a GET request to the server will return all data available from the bulb (include its type, version, mac address, location, name, and if it is online).

```
GET /WebMagicHome///api///AB004/GetMyBindModuleItems?CDPID=Rayn01 HTTP/1.1
Host: ryan.magichue.net
Accept: */*
Cookie:
.ASPXAUTH=8B8CEDE871EE5E49FC85CA12EE9B9AB0E06CFD7D6A695DDB10794BDA590B783AE17C7725D
3DDD971D17A4ABD8D2223401AEE96FCB6FB99B9D84E3090ED4B9091E3DE8E652D2F2479000C61100450
8312A32197CE45466507559EA44B6C8BF72A0480EE77759A0CC2C28C908235FE7C528DE5C1D1BFFBABA
2A643DC14C1D599DD
User-Agent: LedWiFiProRyan/1.0.9 (iPhone; iOS 11.3; Scale/3.00)
Accept-Language: en-GB;q=1, en-SG;q=0.9, zh-Hans-SG;q=0.8, ko-SG;q=0.7
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 04 May 2018 03:47:52 GMT
Content-Length: 240

{"OK":true,"err_code":0,"err_msg":"","Result":[{"DeviceType":68,"LedVersionNum":
8,"ModuleID":"AK001-ZJ200","MacAddress":"600194912715","TimeZoneID":"America/
Chicago","DSToffset":3600,"RawOffset":0,"DeviceName":"singapore","IsOnline":true}]}
```

Fig 8: GET request with the ASPXAUTH cookie and the server response

Additionally, POST requests with this data payload:

{"Password":"57fa01ab27e37922c3faae5343192bf1","UserID":"realemailhere@gmail.com","Devices":[{"DeviceName":"gothacked","MacAddress":"600194912715"}]}

allow you to change the device name remotely. Like previous requests, the password hash and user id are unencrypted and visible, if packets are captured using Wireshark or similar tool.

However, the actual account registration (from which we assume the ASPXAUTH cookie is formed from) uses TLSv1.2, so the passwords are secure and plaintext passwords are not used.

749	52.111996	47.88.101.237	192.168.1.14	TLSv1.2	225	Server Hello, Certificate, Certificate Status, Server Key Exchange, Server Hello Done
751	52.211201	192.168.1.14	47.88.101.237	TLSv1.2	248	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
753	52.268612	47.88.101.237	192.168.1.14	TLSv1.2	173	Change Cipher Spec, Encrypted Handshake Message
755	52.269452	192.168.1.14	47.88.101.237	TLSv1.2	503	Application Data
756	52.269681	192.168.1.14	47.88.101.237	TLSv1.2	375	Application Data
758	52.398147	47.88.101.237	192.168.1.14	TLSv1.2	455	Application Data
926	82.895682	192.168.1.14	47.88.101.237	TLSv1.2	151	Encrypted Alert

Fig 9: Encrypted communication when creating a new user account

3.5 Bulb setup

We also investigated how the bulb is setup. Upon hardware reset (turn off rapidly 4 times), the bulb creates its own unsecured wifi network, which the app needs to connect to. On this network, the same port 5577 is open, and TCP connections that send the hex commands are able to control the light. However, to actually connect the lightbulb to another network, commands are issued through port 48899.

The app opens a UDP connection to port 48899, and sends a series of commands starting with *AT+*

2300	54.491483	10.10.123.5	10.10.123.3	UDP	59	56256 → 48899 Len=17
2301	54.522469	10.10.123.3	10.10.123.5	UDP	78	48899 → 56256 Len=36
2302	54.528104	10.10.123.5	10.10.123.3	UDP	59	49239 → 48899 Len=17
2303	54.532261	10.10.123.3	10.10.123.5	UDP	78	48899 → 49239 Len=36
2304	54.532518	10.10.123.5	10.10.123.3	UDP	45	49239 → 48899 Len=3
2305	54.532529	10.10.123.5	10.10.123.3	UDP	51	49239 → 48899 Len=9
2306	54.537804	10.10.123.3	10.10.123.5	UDP	79	48899 → 49239 Len=37
2307	54.557581	10.10.123.5	10.10.123.3	UDP	47	49239 → 48899 Len=5

Fig 10: UDP communication between the app and the bulb on the bulb's network

Here, the app first sends *AT+WSCAN* which appears to be wifi networks that the bulb can detect shown below on wireshark. The channel, SSID, BSSID, security, and indicator are included.

```
AT+WSCAN
+ok=
Ch,SSID,BSSID,Security,Indicator
1,Da Ji Ji,D4:6E:0E:87:8D:4E,WPAPSKWPA2PSK/AES,44
1,xfinitywifi,1A:05:01:FE:2B:A8,OPEN,60
1,CXNK0049CD1B,CC:BE:59:84:25:40,WPAPSKWPA2PSK/AES,54
1,TP-LINK_F5B7A0,C4:E9:84:F5:B7:A0,WPAPSKWPA2PSK/AES,48
1,ANetwork,0C:51:01:EA:F0:88,WPAPSKWPA2PSK/AES,42
1,NETGEAR15,C0:FF:D4:7E:39:A8,WPA2PSK/AES,48
1,CXNK0049CD4A,CC:BE:59:84:26:E7,WPAPSKWPA2PSK/AES,54
1,Letsplaybag,10:05:01:FE:2B:A8,WPAPSKWPA2PSK/AES,55
1,Elq,F8:32:E4:52:C4:E8,WPA2PSK/AES,40
3,GingerJin_UIUC,08:02:8E:DA:20:14,WPA2PSK/AES,57
3,Sexy Boula,B0:4E:26:39:50:9C,WPAPSKWPA2PSK/AES,48
3,GoIrish,C4:3D:C7:93:E8:C2,WPA2PSK/AES,48
3,shire,08:02:8E:D0:AF:B2,WPA2PSK/AES,43
3,newbs,62:38:E0:0A:0F:D6,WPA2PSK/AES,41
1,TP-LINK_6AD8,18:A6:F7:B0:6A:D8,WPA2PSK/AES,47
4,NETGEAR36,9C:3D:CF:D1:38:41,WPA2PSK/AES,40
5,TP-Link_CC2E,70:4F:57:7D:CC:2E,WPAPSKWPA2PSK/AES,47
5,Skyrim,94:77:2B:92:F8:04,WPAPSKWPA2PSK/AES,41
6,aptsjtu,C0:C1:C0:76:23:73,WPAPSK/AES,67
```

Fig 11: Communication between the app (red) and the bulb (blue) with a selection of SSIDs

Next, via user input, the app sends the wifi network and password unencrypted back to the bulb for it to connect to the network. The app also sends initialization data back to the bulb for it to be able to communicate with the remote server. The bulb is told to connect to ryan.magichue.net on port 8805 via TCP.

```
AT+WSSSID=NETGEAR97
+ok=AT+WSKEY=WPA2PSK,AES,0123456789
+ok=AT+WMODE=STA
+ok=AT+SOCKB=TCP,8805,ryan.magichue.net
+ok=AT+Z
+ok=AT+Z
+ok=
```

Fig 12: Initialization data sent by the app (red) and acks from the bulb (blue)

Commands from the app use *AT+*[command], and the bulb always responds with *+ok=* as an acknowledgement.

3.6 Exploits

On the local network, any TCP connection to a Magic Bulb on port 5577 with a valid command will be accepted. There is no authentication whatsoever, and any and all commands are accepted by the light bulb. Not many products or services use port 5577, so simply scanning a subnet with devices with that port open, and sending a hex command, and receiving an ack back will confirm that a bulb is on that IP. Proof-of-concept code written in Python is here**.** The program has options of different strobe speeds and colors, the ability to specify an IP or IPs, or will scan all ips on the subnet for light bulbs. There is also an option to simply change the color on the bulb. Additionally, the code will send the command to turn on the light on periodically, so the only way a victim can stop the light is to unplug the power.

A cinematic video demo is shown here.

Remote control of the light bulb is possible if the ASPXAUTH cookie is obtained. This is possible though interception of HTTP requests to the remote server, as it is unencrypted[7]. Once this cookie is obtained, the data about the bulb can be obtained by sending a get request to the server, which returns json data containing bulb information as described in Figure 8, which is also unencrypted. From our testing, the cookie seems to last for at least a couple of days, and can be reused indefinitely during that time. Proof-of-Concept code to demonstrate this remote exploitation is here. The program performs the same actions as the local version, except this code can be run anywhere with internet, and goes through the remote server.

A longer not-as-fun, non-cinematic demo is here

3.7 Other Exploits

More can be done by intercepting the POST requests to the remote server. Capturing the (unencrypted) requests with the user and password also allows the bulb name to be changed easily, by sending the POST request with data in the format described at the end of Section 3.4. Demo here! The consequences of this are that the bulb can be changed inside a network (containing sensitive data) and then the data can be accessed outside of the network with possession of the cookie, which is easily obtained.

Also, the bulb can be easily made to connect to another wifi network. By resetting the bulb physically, an attacker can connect the bulb to any network, by establishing a UDP connection on port 48899 and sending the bulb the network SSID and password as shown in Figure 12.

[7] To be able to read these packets, you still need to decrypt the WEP/WPA/WPA2 packets

4 Future possibilities

Like the Hue, we could break the bulb open and try to find vulnerabilities in the hardware, and obtaining access to the firmware to get a better understanding of how the bulb operates, and the protocols and commands that it uses.

Another possibility is to figure out how the .ASPXAUTH cookie is created, and if it can be replicated or unencrypted, because the cookie seems to be the key to how the bulb authenticates with the remote server.