# An Experimental Study of Security and Privacy Risks with Emerging Household Appliances (Position Paper)

Sukhvir Notra[†], Muhammad Siddiqi[†], Hassan Habibi Gharakheili[†], Vijay Sivaraman[†*], Roksana Boreli[*†]
† School of Electrical Engineering & Telecommunications, UNSW, Sydney, Australia.
∗ Networks Research Group, NICTA, Sydney, Australia.
Emails: {*sukhvir.notra@student., m.siddiqi@, h.habibi@, vijay@*}*unsw.edu.au,* {*roksana.boreli@nicta.com.au*}

***Abstract*—Smart household appliances, ranging from light-bulbs and door-locks to power switches and smoke-alarms, are rapidly emerging in the marketplace, with predictions that over 2 billion devices will be installed within the next four years. However, security implementations vary widely across these devices, while privacy implications are unclear to users. In this paper we dissect the behavior of a few household devices, specifically the Phillips Hue light-bulb, the Belkin WeMo power switch, and the Nest smoke-alarm, and highlight the ease with which security and privacy can be compromised. We then propose a new solution to protect such devices by restricting access at the network-level. Our solution does not require changes from device manufacturers, reduces burden on the end-users, and allows security to be offered as an overlay service by the ISP or from a specialist provider in the cloud.**

## I. Introduction

The home is becoming increasingly "smart", driven by emergence of Internet-connected appliances, referred to as the Internet-of-Things (IoT). This enables consumers to remotely monitor and manage their home environment [1] – we can lock/unlock doors from miles away, smoke alarms can alert your mobile phone when a fire is detected, and lighting systems can be controlled remotely. Surveys in the US [2] indicate personal or family safety, property protection, lighting/energy management, and pet monitoring as top motivations for use of such devices, with 51% of those surveyed willing to pay in excess of $500 for a fully-equipped smart-home.

IoT devices are increasingly equipped with sensors (cameras, microphones, motion detectors, etc.) and actuators (e.g., lights, speakers, locks), which raise privacy and security concerns at an unprecedented scale. For example, cameras and microphones built into surveillance equipment can be used by hackers to spy on family activities. There are reported cases [3] of hackers breaking into Internet-connected baby monitors to speak obscenities and intrude on the family. Some IoT applications are tightly linked to sensitive infrastructures and strategic services such as water/electricity distribution and surveillance of the assets. Other applications handle sensitive information about people, such as their location and movements, or their health and purchasing preferences. A smart home lets a legitimate user control all of his/her devices with ease. Unfortunately, it also provides a platform for the attackers to hack into the home network, and remotely control various automated systems. In this paper we explore the network behaviour of three IoT devices including the Phillips Hue light-bulb, Belkin WeMo power switch, and Nest smoke-alarm. Based on experimental evaluation, we highlight and demonstrate the lack of encryption, appropriate authentication, message integrity checks and also the privacy implications for these IoTs.

Existing solutions attempt to improve security of each device by embedding it into IoTs, which requires changing existing communication mechanisms and protocols. These embedded solutions aim at enhancing encryption, authentication, and key management. However, implementing such solutions may not be feasible on a large scale, with potentially high additional costs compared to the cost of IoT devices, and with resource intensive overheads in terms of computational processing and power consumption. Moreover, enhancing the products in this way could require redesign of physical structure, which may not be an option for some IoTs. This leads us to believe that a network-level approach can be more efficient and acceptable implementation choice. We advocate dynamic access control rules and policies, to augment the device-level security provided by the manufacturers. Also, due to the wide range of IoT devices and communication protocols, we believe that the security needs to be outsourced as a service (i.e. SaaS) to a "specialized" provider. The SaaS provider would then maintain a database of specific rules/policies for IoTs. Whenever a new IoT device is connected to the network, the network user, administrator or even the ISP could query protection methods from the SaaS provider. In this paper, we provide the following contributions.

- We study the network activity of three devices including Phillips Hue light-bulb, the Belkin WeMo power switch, and the Nest smoke-alarm in our lab, and demonstrate the ease with which security and privacy can be compromised for these IoTs, and
- We develop a network-level access control solution that further strengthens any security mechanisms that may exist in the IoT device, and demonstrate its value for the three devices considered.

The remainder of the paper is organized as follows: §II summarizes the relevant prior work. In §III we demonstrate the security and privacy vulnerabilities of selected IoT devices. We describe our solution and experimental results in §IV, and the paper concludes in §V.

978-1-4799-5890-0/14/$31.00 ©2014 IEEE

## II. Background and Related Work

Security and privacy of IoT is in its infancy and most of the research work is on understanding and identifying the security and privacy threats and searching the existing suite of security techniques to look for appropriate solutions [4]. Most of the researchers advocate device embedded security architectures. Some researchers propose to standardize IoT communication protocols, for example, DTLS optimization has been suggested to secure the IoT data exchange [5]. Implementation of IEEE 802.15.4-compliant link layer security procedures is advised in [6] and a lightweight encryption/decryption method for ID authentication among nodes in sensor layer has been presented in [7]. Concepts from Artificial Immune System (AIS) have been imported to detect attacks on IoT and have used those concepts to develop an IoT intrusion detection system with dynamic defense [8], [9]. VIRTUS [10], a middleware solution for management of applications in IoT environments adopts open standards such as XMPP and OSGi. Our demonstration of risks associated with IoT devices in this paper is inspired by [11] and [12]. However, they only present some vulnerabilities of IoT devices and develop attacks accordingly and the solution is not proposed.

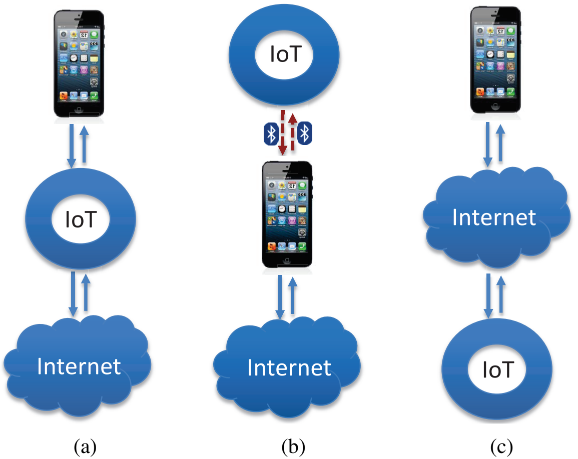
Location privacy system based on the concept of Virtual ID, that only exposes location in critical situations, is proposed in [13]. Location privacy in the context of requested network-based services with the ability of automated privacy choices has been developed in [14]. Optimised implementation of ECDSA and token based access to CoAP resources have been used to develop an access control mechanism in [15].

Most of the prior works either partly address the problem or propose high-level security architectures involving changes to the way IoT devices are currently designed and communicating. With hundreds of IoT device manufacturers, it is almost impossible to come up with a device embedded security solution that caters to all the security and privacy threats for a variety of IoT devices with varying capabilities. Also due to the miniature size of many IoT devices with limited computing capabilities and power resources make it impossible to apply extensive computing-rich security algorithms. We also believe that device embedded solutions requires all manufacturers to be on board which is a hard task. Our approach however assures security by applying a dynamic set of network-level rules to limit IoT access to/from legitimate entities (i.e. Apps, servers, and users).

## III. Demonstrating Security and Privacy Risks

In this section we present our findings about the vulnerabilities (ranging from encryption, authentication, access control, message integrity and suspicious private data transfer), associated with selected IoT devices. We have selected three devices that are arguably the ones most popularly purchased and deployed today: the **Nest Smoke-Alarm**, the **Hue light-bulb**, and the **WeMo Motion Switch**. For all these devices installed in our lab, we have captured the network activity using the Wireshark packet analyzer.

### A. Operational Models



(a) (b) (c)

Fig. 1. Operational model: (a) Direct, (b) Transit, (c) External server

For all IoT devices, we have identified three main models of communication between the user, IoT and the cloud-based server maintained by the manufacturer, as shown in Fig. 1. "Direct" access model allows the user to directly communicate with the IoT device (e.g. Philips Hue light-bulb and WeMo motion/switch) via the mobile App, then as depicted in Fig. 1(a), the IoT device updates the server of the current status. It is also possible to control the IoT device via the web portal provided by the manufacturer. "Transit" model is mainly used for IoT devices those do not have the capability to directly exchange data with the Internet-based server. These type of devices (e.g. Fitbit fitness tracker) do not have Wi-Fi interface and generally use other means of communication such as Bluetooth and Near field communication (NFC). In order to communicate with the server to retrieve the required data, the user's phone is used as a relay/bridge for data exchange, as shown in Fig. 1(b). We note that this model is not considered in the current study. Lastly, in the "External" model, the user has no direct interaction with the IoT device. Fig. 1(c) shows that the IoT device (e.g. Nest Protect smoke alarm) communicates directly with the external sever, and the only way for the user to retrieve relevant data, such as current status, is via this external server.

### B. Nest Smoke-Alarm

The first IoT device we used in our lab was a Nest Protect smoke and carbon monoxide alarm. It communicates with user through a mixture of verbal warnings, beeps, LED lights and text alerts on the mobile app. It is equipped with a photoelectric smoke sensor, a carbon monoxide sensor, a speaker, and four sensors that detect motion, light, and heat. It is smart enough to know when the user is in the same room or if he/she has turned on the lights. These kind of capabilities would raise a privacy concern for the users who may feel that they are being monitored and tracked potentially. Using a 3-day long traffic snapshot of our smoke-alarm, we have found that all data exchange over the network is encrypted. Analyzing this captured data shows that the Nest device is never contacted by any external servers to retrieve information. All traffic to external servers is initiated by smoke alarm sensor once a day, during normal situation (i.e. no fire). However, in case of emergency, the sensor immediately sends messages to a set of external servers and then the user's app is notified as well.

As depicted in Fig. 2, we have examined the flow of data for Nest network activity by inspecting wireshark

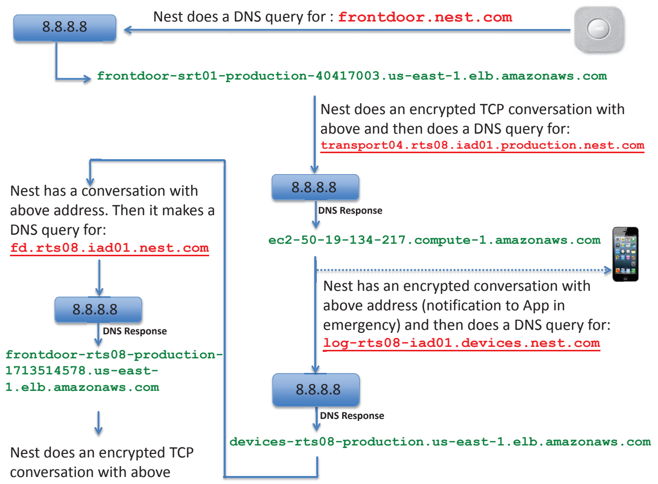

8.8.8.8
Nest does a DNS query for : frontdoor.nest.com
frontdoor-srt01-production-40417003.us-east-1.elb.amazonaws.com
Nest does an encrypted TCP conversation with above and then does a DNS query for:
transport04.rts08.iad01.production.nest.com
8.8.8.8
DNS Response
ec2-50-19-134-217.compute-1.amazonaws.com
Nest has an encrypted conversation with above address (notification to App in emergency) and then does a DNS query for:
log-rts08-iad01.devices.nest.com
8.8.8.8
DNS Response
devices-rts08-production.us-east-1.elb.amazonaws.com
Nest has a conversation with above address. Then it makes a DNS query for:
fd.rts08.iad01.nest.com
8.8.8.8
DNS Response
frontdoor-rts08-production-1713514578.us-east-1.elb.amazonaws.com
Nest does an encrypted TCP conversation with above

Fig. 2. Nest communication process

packet capture. The smoke-alarm device authenticates itself using its "OAuth2" token. We believe authentication occurs during first conversation with `frontdoor.nest.com`. After this stage, the Nest device talks to another server at `transport04.rts08.iad01.production.nest.com` which we shall refer to as "Notification server". This server is responsible for sending notifications to user's App in case of emergency. We corroborated this fact by conducting packet capture of Nest mobile App traffic (using **tPacketCapture** on Android), when the App is launched. We found that, the size of data transfer to this server was less than 1 KB. After concluding this conversation, Nest device does a DNS query for `log-rts08-iad01.devices.nest.com`. As suggested by the name of this server it seems to be a "log server" which stores all data collected by the IoT device. We also found that the size of this conversation was about 20 KB on average, which is another indication of the fact that this is the actual data storing server. Lastly, Nest sensor concludes the daily sync process by contacting to `fd.rts08.iad01.nest.com`. Relying on OAuth 2.0 protocol flow, we think this server could be potentially used to obtain a new authentication token for next day sync process. In order to inspect the Nest activity in case of emergency, we conducted a fire test in our lab. We found that the smoke-alarm followed the same process as during unprompted daily sync activity, except the last conversation (with `fd.rts08.iad01.nest.com`). We think it is because Nest retains the previously allocated token for 24 hours and reuses it in case of emergency.

All things considered, Nest is actually quite a secure product. All the communication from the smoke-alarm is encrypted. Security is further enhanced by Single-Sign-On service with OAuth2. This makes it very hard for someone to eavesdrop and extract information about a user from the communication between Nest sensor and servers.

In terms of privacy, there are a few concerns. The above mentioned behaviour seems to suggest that the notification server deals with the basic smoke-alarm functionality, i.e. obtaining smoke alarm status from the device and notifying the user on their mobile app. This leaves open the question of the purpose of the log server, particularly since that communication is an order of magnitude larger in volume than the alarm functionality. This conversation also occurs after the notification which raises concerns about its necessity. While we cannot comment (due to encrypted conversations) on the content of this data transfer, it has the potential to carry private user information that is gained from motion and light sensors.

### C. Hue Light-Bulbs

We now explore the network activity of Hue light-bulbs. The Philips Hue Connected bulb allows the user to wirelessly control the lighting system in the home. The user can adjust intensity, set custom colors, color combinations, and schedules via Philips Hue App for Android and iOS. Philips hue is also very customizable and and 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`) service. For example, the user can setup a recipe to turn the light bulbs to colors from a photo (s)he has been tagged in or turn the light off when user's phone is out of the WiFi range.

We have installed the Philips hue starter package consisting of three LED light bulbs and an ethernet enabled bridge. The bridge acts as a controller for all the bulbs. All communications from the App are directed towards the bridge. Bridge then sends appropriate signals to the bulbs for a desired change, using ZigBee Light link protocol. It is also possible to control the lightbulbs from the `meethue.com` web-site when not home. However, the users need to associate their bridge with a specific account first.

When the Hue App is launched, it first generates a hash-like username (in our case: `v7Le0FDyDCh3NLcE`) and then checks to see if the bridge has its username "whitelisted", by issuing the following `GET` request:

```
GET /api/v7Le0FDyDCh3NLcE HTTP/1.1
Host: 129.94.5.95
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
```

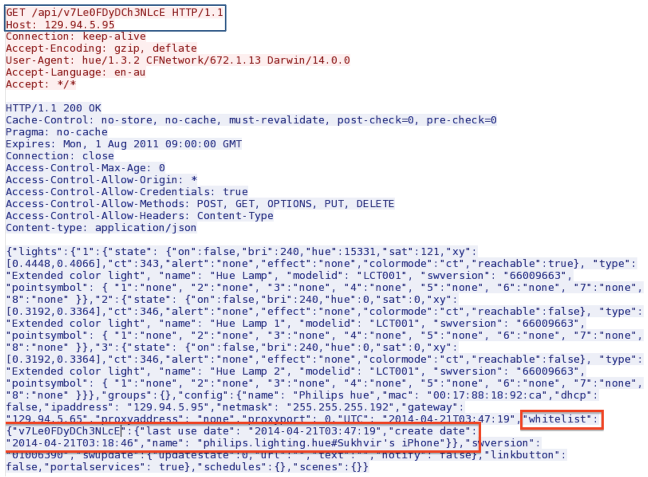
Note that the above `129.94.5.95` is the bridge's IP address. Here is the response from the bridge to the App, if it is not authorized:

```
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 while continuously sending `GET` requests. Once the button is pressed, the bridge response changes as following:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon,21 April 2014 23:00:00 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":"v7Le0FDyDCh3NLcE"]
```

At this point the app (i.e. username) is added to the whitelist maintained by the bridge. In order to control the lights, the app makes PUT request with appropriate message to the bridge, using the whitelisted username. Bridge then replies back with HTTP/1.1 200 OK response to indicate a successful operation.



```
GET /api/v7Le0FDyDCh3NLcE HTTP/1.1
Host: 129.94.5.95
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: hue/1.3.2 CFNetwork/672.1.13 Darwin/14.0.0
Accept-Language: en-au
Accept: */*

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

{"lights":{"1":{"state": {"on":false,"bri":240,"hue":15331,"sat":121,"xy":
[0.4448,0.4066],"ct":343,"alert":"none","effect":"none","colormode":"ct","reachable":true}, "type":
"Extended color light", "name": "Hue Lamp", "modelid": "LCT001", "swversion": "66009663",
"pointsymbol": { "1":"none", "2":"none", "3":"none", "4":"none", "5":"none", "6":"none", "7":"none",
"8":"none" }},"2":{"state": {"on":false,"bri":240,"hue":0,"sat":0,"xy":
[0.3192,0.3364],"ct":346,"alert":"none","effect":"none","colormode":"ct","reachable":false}, "type":
"Extended color light", "name": "Hue Lamp 1", "modelid": "LCT001", "swversion": "66009663",
"pointsymbol": { "1":"none", "2":"none", "3":"none", "4":"none", "5":"none", "6":"none", "7":"none",
"8":"none" }},"3":{"state": {"on":false,"bri":240,"hue":0,"sat":0,"xy":
[0.3192,0.3364],"ct":346,"alert":"none","effect":"none","colormode":"ct","reachable":false}, "type":
"Extended color light", "name": "Hue Lamp 2", "modelid": "LCT001", "swversion": "66009663",
"pointsymbol": { "1":"none", "2":"none", "3":"none", "4":"none", "5":"none", "6":"none", "7":"none",
"8":"none" }}},"groups":{},"config":{"name": "Philips hue","mac": "00:17:88:18:92:ca","dhcp":
false,"ipaddress": "129.94.5.95","netmask": "255.255.255.192","gateway":
"129.94.5.65","proxyaddress": "none","proxyport": 0,"UTC": "2014-04-21T03:47:19","whitelist":
{"v7Le0FDyDCh3NLcE":{"last use date": "2014-04-21T03:47:19","create date":
"2014-04-21T03:18:46","name": "philips.lighting.hue#Sukhvir's iPhone"}},"swversion":
"01006390","swupdate":{"updatestate":0,"url":"","text":"","notify": false},"linkbutton":
false,"portalservices": true},"schedules":{},"scenes":{}}
```

Fig. 3. Wireshark capture of Philips Hue GET message

Fig. 3 shows an example of our Wireshark capture of GET request/response exchanged between the bridge and app, all in plain text. The top two lines reveal the legitimate username and the bridge's IP address. Also, the whitelist is identified in the response on the bottom in Fig. 3. It also contains current status of lights (brightness/color/hue/alarm status, etc) in plain text giving an attacker great insight into the current state of affairs inside victim's house.

Philips Hue light-bulb was hacked in 2013 by [11] using the above vulnerability. It was shown that the username is not random, rather just MD5 hash of MAC address of the mobile device on which the app is running. Since then, Philips app has changed the mechanism to create the username. However the communication is still in plain text. We have exploited this vulnerability by writing a python script (which we made available on GitHub [16]). To conduct this attack, potential attacker needs to capture traffic (either GET or PUT) between a legitimate user and Hue bridge. This capture can then be used to extract the bridge's IP and the whitelisted username (as shown in Fig. 3). Our code then uses this username to connect to the victim's bridge. It is then possible to retrieve other whitelist usernames by making a GET request using our code. The attacker can also take full control of the bridge by making PUT requests, using our code. For example the following shows a PUT request to turn the lights on with brightness of 254 and color temperature of 50:

```
PUT /api/v7Le0FDyDCh3NLcE/lights/1/state HTTP/1.1
Host: 129.94.5.95
Content-Length: 55
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/2.2.1 CPython/2.7.5 Darwin/13.2.0

{"transitiontime": 0, "on": true, "bri": 254, "ct": 50}
```

The bridge's response is shown below:

```
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: 3600
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE, HEAD
Access-Control-Allow-Headers: Content-Type
Content-type: application/json

[{"success":{"/lights/1/state/transitiontime":0}},
{"success":{"/lights/1/state/on":true}},
{"success":{"/lights/1/state/ct":153}},
{"success":{"/lights/1/state/bri":254}}]
```

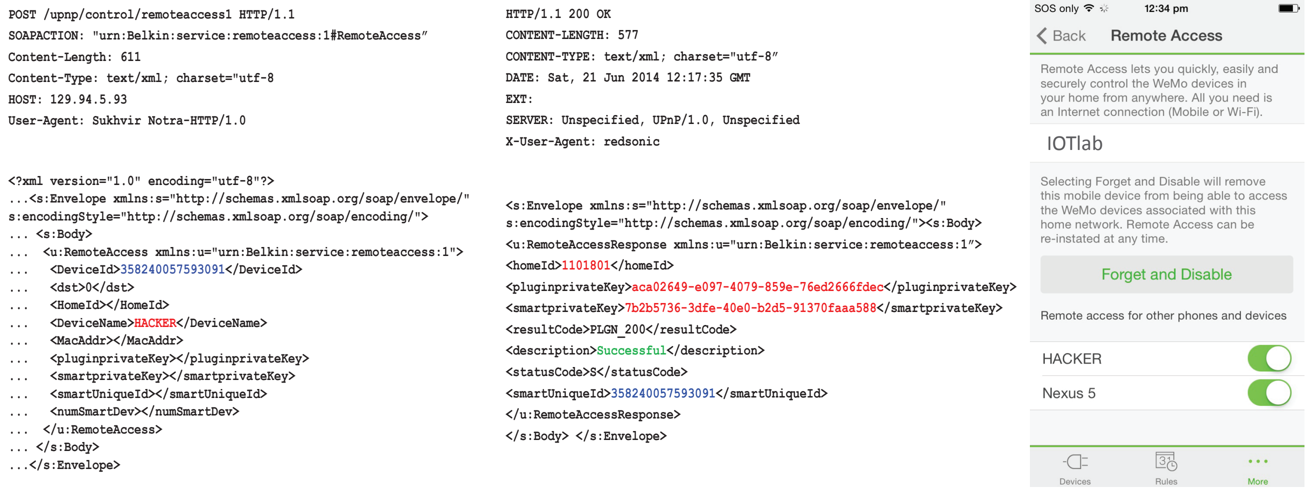
### *D. WeMo Motion Switch*

Lastly, we experimented with the Belkin's WeMo motion sensor and switch kit which is designed to give a wireless control of home appliances to the user. Both of these devices are connected to the Internet and notify the user of any activity via Belkin's WeMo App. WeMo Switch is essentially a power socket and any electrical device such as desk lamps, coffee machines, room heaters etc, can be connected to it. WeMo Motion is an Internet-connected motion sensor which can notify the user if a motion is detected by sending a notification to the App. The user can configure this kit by inserting rules via the App interface regarding a certain action to be taken upon detection of a motion. For example, *Turn the WeMo switch on for 5 minutes when WeMo motion detects a motion.*

When the App is launched, the phone broadcasts a SSDP discovery request to the multicast group 239.255.255.250. If the WeMo switch/motion is in the same LAN as the phone, phone receives a reply from them. If a reply is received, then the phone communicates with the devices directly and then the devices notify the Belkin's cloud-based server (api.xbcs.net) of any change made by the user. If App doesn't receive any SSDP reply, all commands are relayed to the devices via cloud-based server, assuming the "remote access" is enabled on the App by the user.

We have identified a set of security issues with this IoT device listed as following:

1) WeMo devices use a SOAP API for all communications between App and the device. Note that both ends have to be in the same LAN. This SOAP communication is entirely in plain text. This is a vulnerability as an attacker can learn the format of communication and can also capture packets from legitimate user and replay them to the device at a later stage.
2) On observing this communication we discovered that the WeMo devices used no authentication method to ensure that the commands were coming from a legitimate device. This is a major vulnerability and can be easily exploited by an attacker issuing correctly formatted requests to the device.
3) WeMo Motion/Switch list all the services offered by these devices on an exposed xml interface, reported by the SSDP discovery process. Each of these services then list actions (and their arguments) that can be performed by these devices, for example: *Controlling the Switch (on/off)*, *Get the current status of the switch (on/off)*, *Get a list of close by Access Points* (including their encryption mode and signal strengths) to these devices.
4) These devices also have only two fixed transport ports that are used to listen in for user commands (49154 and



```
POST /upnp/control/remoteaccess1 HTTP/1.1
SOAPACTION: "urn:Belkin:service:remoteaccess:1#RemoteAccess"
Content-Length: 611
Content-Type: text/xml; charset="utf-8
HOST: 129.94.5.93
User-Agent: Sukhvir Notra-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>358240057593091</DeviceId>
...   <dst>0</dst>
...   <HomeId></HomeId>
...   <DeviceName>HACKER</DeviceName>
...   <MacAddr></MacAddr>
...   <pluginprivateKey></pluginprivateKey>
...   <smartprivateKey></smartprivateKey>
...   <smartUniqueId></smartUniqueId>
...   <numSmartDev></numSmartDev>
...  </u:RemoteAccess>
... </s:Body>
...</s:Envelope>
```

(a) Request

```
HTTP/1.1 200 OK
CONTENT-LENGTH: 577
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: Sat, 21 Jun 2014 12:17:35 GMT
EXT:
SERVER: Unspecified, UPnP/1.0, Unspecified
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>1101801</homeId>
<pluginprivateKey>aca02649-e097-4079-859e-76ed2666fdec</pluginprivateKey>
<smartprivateKey>7b2b5736-3dfe-40e0-b2d5-91370faaa588</smartprivateKey>
<resultCode>PLGN_200</resultCode>
<description>Successful</description>
<statusCode>S</statusCode>
<smartUniqueId>358240057593091</smartUniqueId>
</u:RemoteAccessResponse>
</s:Body> </s:Envelope>
```

(b) Response

Remote Access lets you quickly, easily and securely control the WeMo devices in your home from anywhere. All you need is an Internet connection (Mobile or Wi-Fi).

IOTlab

Selecting Forget and Disable will remove this mobile device from being able to access the WeMo devices associated with this home network. Remote Access can be re-instated at any time.

Remote access for other phones and devices

HACKER

Nexus 5

(c) App interface

Fig. 4. WeMo switch remote access: (a) request, (b) response, and (c) app interface

49153). Lack of entropy makes this a potential vulnerability, which can be exploited by targeted messages to these devices.

5) These devices use STUN protocol to give Belkin servers the ability to transverse user's NAT and give them the access to these devices. However these devices don't use SSL to connect to the STUN server which leaves the possibility of Man-In-Middle-Attack (MITM) open.

*1) **no-NAT Attack**:* Based on above vulnerabilities, we have written a Python script to emulate an attack. Our attack focuses on vulnerabilities 2, 3, and 4 as listed above. Our code initially conducts an SSDP discovery process to detect the IP address of the WeMo devices and the port they are listening on (49154 or 49153). Once we have the IP and the port, our code then transmits properly formatted SOAP commands to the devices to control the device or query it for any information. These SOAP commands use the actions and arguments as reported by these devices during the SSDP discovery process.

Our code can be found online at Github [16]. The script exploits the fact that WeMo devices don't use any authentication or access control methods to protect their devices. Hence anyone can control these devices with a properly formatted SOAP query. In this attack, we assume the WeMo devices have public IP addresses without NAT enabled.

*2) **Remote Access Attack**:* If WeMo is protected by NAT, it is still possible to take control of these devices remotely. Since no authentication is employed by WeMo devices, anyone with one time access into victim's LAN will then be able to control the devices from anywhere in the world. However, this attack is not scalable. An attacker merely need to download the WeMo App and gain access to victim's LAN. Once inside the LAN, attacker launches their standard WeMo app. App will then conduct a SSDP discovery (per usual) and locate victim's WeMo devices. Attacker can then enable the remote access for his app by going to Setting→Remote Access→Enable Remote Access. Now the attacker's app has remote access to the victim's WeMo devices. The attacker can use this to exploit victim's WeMo devices from anywhere in the world at anytime. This process is also emulated by our code. Attacker just needs to pick a **`DeviceName`** and **`DeviceId`**, then send a properly formatted SOAP request to the devices (Fig. 4(a)). Upon arrival of this request, the WeMo devices reply back with a set of credentials for remote access including **`homeId`**, **`pluginprivateKey`**, **`smartprivateKey`** and **`smartUniqueId`**, as shown in Fig. 4(b).

Now when the attacker tries to control WeMo devices from the app, the illegitimate request is sent to Belkin's servers with proper credentials as provided by WeMo devices. The servers then relay this requests to the IoT devices. This is made possible by the fact that the Belkin's servers are aware of "translated public address" (assuming NAT) for these devices using STUN protocol.

Note that in order to explicitly demonstrate our attack, we chose "HACKER" as our device name as shown in Fig. 4(a). However, in real practice more subtle name such as "My Phone" can be chosen to avoid suspicion from legitimate user, as the app's remote access interface (Fig. 4(c)) is shown to all user's that are linked to that particular device.

Needless to mention, our code [16] stops after it receives the credentials from WeMo devices. However, in order to attack these devices an attacker can contact `http://api.xbcs.net:8443` with a properly formatted request (using these credentials) and the Belkin server will relay the commands to the devices.

## IV. Our Solution: Network-Level Protection

We now propose our solution to protect IoT devices at the network level. In our model, an external entity, called the Security as a Service (SaaS) Provider, maintains a database of access control rules that secure the various IoT products. This rule database is maintained in the cloud and can be updated whenever new vulnerabilities emerge. This service can be queried dynamically via a simple API that takes as input MAC addresses and/or device *friendlyName* and other optional device specific arguments. We have developed a code that represents a SaaS provider's API which is available at [16]. Successful API call returns a set of rules. In what follows we illustrate how this concept can be applied to secure the three devices whose vulnerabilities we have illustrated in the previous section.

### A. Privacy Protection for the Nest Smoke-Alarm

In terms of security, Nest smoke-alarm is a robust product. However, there are potential privacy concerns for the user pertaining to activity monitoring. As mentioned in §III-B, Nest device seems to transfer 20KB of data daily to the log server. While we don't know the contents of this data (because of encryption), the size of this transfer is alarming. In order to safeguard the user's privacy we found that blocking access to the log server (i.e. `log-rts08-iad01.devices.nest.com`) prevents data transfer without jeopardising Nest's ability of notifying user in emergency. We applied the following rule which was obtained from SaaS provider (queried by Nest's MAC address `18-b4-30-xx-xx-xx`) and conducted a fire test:

```
{"firewall": "Enabled",
 "allow": ["174.129.5.148",
           "50.19.134.217",
           "23.23.239.159"],
 "direction": "Outbound",
 "device": ["Nest Labs Inc."]}
```

This rule blocks all outgoing traffic by enabling firewall with the exception of three destinations representing necessary servers (i.e. authentication, notification and token renewal). After applying this rule on the network (i.e. home gateway) and conducting a fire test, we found that the app still received a fire notification. Needless to mention in our experimental setup we did not have any other Nest products installed, to test whether inter-connectivity between multiple Nest products is lost because of our block.

### B. Securing Access to Household Devices

Note that our lab setup is not behind the NAT and the Hue bridge/WeMo switch have public IP addresses. In typical home networks NAT essentially protects these devices from an Internet-based attacker. But these IoTs are vulnerable in enterprise environment such as small-office-small-home (SOHO), campus networks etc. In order to secure the Hue device when the bridge is not behind the NAT server, we applied a dynamic access control rule (as below) on inbound traffic of home gateway, using SaaS rule repository:

```
{"firewall": "Enabled",
 "allow": ["198.142.228.10",
           "162.13.15.30"],
 "direction": "Inbound",
 "device": ["Philips Lighting BV"]}
```

In this case, the allow field contains the permitted source IP addresses (i.e. legitimate user's phone and Philips server). Note that, we have also implemented an Android app which reports the current IP of the legitimate user's phone to our SaaS provider. This enables our SaaS code to update the aforementioned rule and provides dynamic access control. We have tested this simple network-level solution and found that our "attacker" script could not reach the bridge inside the lab (i.e. home) whereas the legitimate user was able to communicate with bridge.

WeMo's switch can be protected in a similar way as Hue's IoT against no-NAT attack. However, the remote access attack can only be avoided by restricting access to the device within the LAN (i.e. home network behind NAT). This could be achieved by isolating WeMo devices using separate VLANs.

## V. Conclusions

As smart homes increasingly adopt IoT devices, security and privacy become important concerns. In this paper we have highlighted real vulnerabilities in selected popular devices used in today's home environment, and explored a solution that protects them at the network-level. We undertook extensive analysis of the captured packet traces to identify current vulnerabilities, and developed scripts that can automate the attacks at scale. We have then developed a solution that allows an external entity, such as the ISP or a SaaS provider, to install appropriate access control rules in the network to protect these IoT devices against external threats. We believe our tool is the first step towards more sophisticated and comprehensive IoT security and privacy assurance, and can be generalized and extended to other devices in the future.

## References

[1] C. Wan and D. Low, "Capturing Next Generation Smart Home Users with Digital Home," Huawei, White Paper, Jun. 2013.

[2] iControl. (2014) 2014 State of the Smart Home.

[3] abcNEWS. (2013) Baby Monitor Hacking Alarms Houston Parents. http://goo.gl/LPuJzg.

[4] H. Suo, J. Wan, C. Zou, and J. Liu, "Security in the internet of things: A review," in *Proc. of Computer Science and Electronics Engineering (ICCSEE)*, March 2012.

[5] S. Keoh, S. Kumar, and H. Tschofenig, "Securing the internet of things: A standardization perspective," *Internet of Things Journal, IEEE*, vol. 1, no. 3, pp. 265–275, June 2014.

[6] D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, and M. Rossi, "Low power link layer security for iot: Implementation and performance analysis," in *Proc. of Wireless Communications and Mobile Computing Conference (IWCMC)*, July 2013.

[7] Q. Wen, X. Dong, and R. Zhang, "Application of dynamic variable cipher security certificate in internet of things," in *Proc. of Cloud Computing and Intelligent Systems (CCIS)*, Oct 2012.

[8] C. Liu, J. Yang, Y. Zhang, R. Chen, and J. Zeng, "Research on immunity-based intrusion detection technology for the internet of things," in *Proc. of Natural Computation (ICNC)*, July 2011.

[9] C. Liu, Y. Zhang, and H. Zhang, "A novel approach to iot security based on immunology," in *Proc. of Computational Intelligence and Security (CIS)*, Dec 2013.

[10] D. Conzon, T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M. Spirito, "The virtus middleware: An xmpp based architecture for secure iot communications," in *Proc. of Computer Communications and Networks (ICCCN)*, July 2012.

[11] N. Dhanjani. (2013) Hacking Lightbulbs. http://goo.gl/RY252I.

[12] N. Dhanjani. (2013) Reconsidering the perimeter security argument. http://goo.gl/ukUELD.

[13] C. Hu, J. Zhang, and Q. Wen, "An identity-based personal location system with protected privacy in iot," in *Proc. of IEEE Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on*, Oct 2011.

[14] M. Elkhodr, S. Shahrestani, and H. Cheung, "A contextual-adaptive location disclosure agent for general devices in the internet of things," in *Proc. IEEE LCN Workshops*, Oct 2013.

[15] A. Skarmeta, J. Hernandez-Ramos, and M. Moreno, "A decentralized approach for security and privacy challenges in the internet of things," in *Proc. of Internet of Things (WF-IoT)*, March 2014.

[16] S. Notra. (2014) UNSW IoT. https://github.com/sukhvir-notra/UNSW-IoT.