Using Software Defined Radio to Attack "Smart Home" Systems

GIAC (GCIH) Gold Certification

Author: Florian Eichelberger, florian.eichelberger@cognosec.com
Advisor: Dr. Kees Leune

Accepted: Month 27th 2015
(Date your final draft is accepted by your advisor)

Template Version September 2014

Abstract

The Internet of Things (or IoT) is an emerging trend of which *Smart Homes* is a subset. IoT involves the integration of digital and wireless technologies in physical objects and systems, especially those historically unconnected. Home Automation systems or "Smart Homes" have been an emerging trend, with products only recently hitting the mass market and being affordable. Out of a fear of reduced usability, or breaking backwards compatibility, security is often neglected, or added as an afterthought. With the emergence of consumer-ready, programmable radio systems and low-cost devices with sufficient computational power, the field of Software-defined-radio (SDR) is experiencing rapid growth. Researchers can evaluate and attack all kinds of wireless systems without having to invest in expensive hardware or circuitry-making equipment, making attacks on these systems significantly easier. This paper describes several plausible attacks on "Smart-Home" systems using SDR platforms. In particular, the implementation of ZigBee, a communications protocol, in recent products is exemplarily examined and a software framework based on existing tools for attacks and audit is proposed and implemented in a proof-of-concept code.

1. Objective

The objective of this paper is to describe several plausible attacks that target "Smart-Home" systems using SDR platforms. In particular, the implementation of ZigBee in recent products is exemplarily examined and a software framework based on existing tools for practical, readily useable and hardware independent, attacks and audit is proposed and implemented in a Proof-of-concept code. This paper attempts to improve on the current status of software implementations that normally focus on one point (i.e. sniffing of wireless traffic). The proposed proof-of-concept code will implement command injections for the popular “scapy-radio” platform.

2. Introduction

The Internet of Things (IoT) is considered to be the next phase of the Internet revolution - linking real objects in the real world to the virtual world and enabling anytime, anyplace and anything communication. (Santucci 2010, p. 11) The fact that the IoT was the main topic of Samsung CEO’s and president BK Yoon’s keynote at the Consumer Electronic Show (CES) 2015 in Las Vegas, illustrates the position of the IoT as one of the most emerging markets and trends.

One of the main applications of the IoT, besides the iconic web-connected refrigerator (Bennet 2009), is the area of home-automation and “Smart Homes”. Even though these prophecies have to be taken with more than a grain of salt, Gartner expects a typical family home to own and use around 500 “smart” devices by the year 2022 (Gartner 2014b). Since communicating over wireless channels seems to be an obvious choice, various security issues arise. Some of these issues are new, but most have actually been around for a long time. A desired short time-to-market, as well as backward compatibility and future proofing considerations, lead to the persistence of known problems.

This paper and the proposed solution examines one of the most widespread technologies currently used for the IoT, ZigBee, and provides an extension to the scapy-radio framework. This will enable security researchers to not only simply sniff traffic, but also to inject traffic directly and seamlessly into the communication protocols, instead of simply sending packets and therefore giving the researcher the ability to actively interact with devices being tested and evaluated.

Prior and Related Work

ZigBee has been around for some time now, and a significant body of related works have been published. One of the more comprehensive publications on ZigBee was the publication in 2009 by S. Farahani (Farahani 2009). Travis Goodspeed (2009) also published papers on the insecurities of common ZigBee hardware and proposed an approach how to attack similar devices.

As SDR was gaining momentum, ZigBee related software was published by the scapy-radio project, the killerbee project as well as by T.Schmid in 2006. The objective of this software was to enable the easy usage of SDR hardware. Those projects and their software are described later on in this paper.
Some papers on attacking ZigBee focus only on one topic, like jamming (e.g. Melgares 2011) or sniffing and replaying beacon packets and performing range measurements (e.g. Dalrymple 2014).

There are numerous other papers on general attack vectors on ZigBee that could be applied to other networks and systems as well (e.g. Markert, Massoth, Fischer-Hellmann, Furnell & Bolan 2011). However, only a small amount of work has been published on practical, useable ZigBee packet injection software using general SDR hardware and open-source software, the most useable software in this regard is the available killerbee framework, which is limited to 3 devices that are supported
(Wright, Melgares 2009)

Motivation

During the security assessment of smart-home devices used for home-automation it turned out that some of these devices used encrypted communications secured by SSL certificates preinstalled by the device vendor that were not properly validated. Consequently, they were vulnerable to man-in-the-middle attacks. Finding this possible attack vector lead to further investigations of how these systems can be exploited.

SDR hardware was easily available but proper encryption support, needed for some attacks using seamless packet injection was missing in scapy-radio software. Extending scapy-radio by implementing encryption was a goal to be pursued in order to provide extended possibilities for security tests.

History of the "Internet of Things"

The foundations of the Internet of Things reach back to visionaries like Nikola Tesla (1926), when he said *"When wireless* is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole.........and the instruments through which we shall be able to do this will be amazingly simple compared with our present telephone. A man will be able to carry one in his vest pocket."* (Kennedy 1926, p. 163)

Followed by Marshall McLuhan in 1964 (McLuhan 1966, p. 57) and Karl Steinbuch in 1966 (Steinbuch 1966, p. 199), general ideas of ubiquitous or interwoven computing are already postulated. The term "Internet of Things" was coined by Kevin Ashton while working for Proctor & Gamble in 1999 as a title of one of his presentations, whereas in this early case, the IoT was mostly related to the tracking of RFID objects. (Ashton 2009)

Definition

The term "Internet of Things" is not commonly defined. It is used as an umbrella keyword for covering aspects of extending the Internet to the real world. "Smart things" are defined in the literature as entities that: (Miorandi 2012, p. 1498)

- Are physical objects with physical features (e.g. size, shape)
- Have communication functionalities
- Are assigned a unique identification token
- Are human and machine addressable
- Posses basic computing power
- May allow mostly physical interaction and offer sensing capabilities

On the economic impact, Gartner foresees around 26 billion devices in 2020 generating around 300 billion USD in revenue for service and device suppliers. (Gartner 2014a)

3. Smart Home

Even though there are many fields of application for "Smart things" and the IoT, this paper is focused on one common application that most Internet users will likely encounter in their personal life. The "Smart Home", next to the closely related smart metering, or automotive systems, smart health and independent living products, is one of main areas of application of this technology.

The concept of a "Smart Home" dates back to the 1950s / 1960s. (Ross 1958, p. 197) and the concept of "home-automation" or "Smart Home" is often used synonymously (Lutolf 1992, p. 277) and for the sake of this paper will be used in this way.

The requirements of wireless "smart devices", as listed below, reflect the aspects that need to be taken into consideration in defining their protocol and design (Xing, Srinivasan, Rivera, Li & Cheng 2010, p. 252)

- Limited resources, limits in energy and computational power limit the usage of highly complex security-aware hardware
- Limited Reliability, based on the limited resources
- Dynamic Topology, by using wireless networking, the probability of network outage or parts being unreachable increases.
- Large Number of Sensors, a large number of sensors need to be manageable and preferable be cheap enough to produce enough synergies in the data they collect.
- Centralized Processing Hub, a central system is collecting sensor information and sending commands to specific or all sensors based on defined criteria.

4. ZigBee ver.02

One of the most widespread technologies used in smart things is the ZigBee protocol. In order to be able to understand the reason why ZigBee was developed and the security issues within the protocol, this paragraph provides a short overview of the ZigBee protocol.

The ZigBee protocol is a superset of the IEEE 802.15.4 RF standard that was ratified by the US Institute of Electrical and Electronics Engineers in the summer of 2003 and then endorsed by the ZigBee Alliance (Poole 2004, p. 44). The ZigBee Alliance is a consortium that was founded in October 2002 by Philips, Motorola, Honeywell, Invensys and Mitsubishi. By 2003 it already had 25 members (Evans-Pughe 2003, p.28) and today has about 400 members as shown on http://www.zigbee.org .

ZigBee was created as Bluetooth technology and was found unsuitable for building automation and industrial controls due to its limited number of possible nodes (Rathnayaka, Potdar. Kuruppu 2011 p. 80) and the lower power consumption of ZigBee, leading to longer usability of the devices.. (Obaid, Rashed, Abou-Elnour, Rehan, Saleh, Tarique 2014 p. 126)

The ZigBee Protocol operates in the 2.4Ghz band or the 868 / 915 MHz Band. With an operational range of 10-75 m, it is well on-par or above Bluetooth specifications but with only the speed of 250 kbps it was below the Bluetooth data rate of 1 Mbps. One of the critical advantages of ZigBee v.01 over Bluetooth 1.2 is the much lower latency. ZigBee devices have a latency of 15 milliseconds in a state with all circuitry switched off (apart from a clock running at 32kHz) to wake up and get a packet across a network. A Bluetooth device in a similar state would take around three seconds to do the same. (Evans-Pughe 2003, p. 30f). ZigBee features three network topologies, a star, a mesh and a cluster tree network, featuring up to 65.000 devices per network. (Poole 2004, p. 45)

Currently there are three versions of the ZigBee Protocol (ver. 01 (2004), ver.02 (2006), and ver.02 (2007)), with the current one only providing compatibility with the 2006 (ver.2) version, and not with the original version.

The ZigBee stack consists of 4 layers: (ZigBee Alliance 2008, p. 35)

- Physical Layer

- Medium Access Control Layer
- Network Layer
- Application Layer

Figure 1 shows an overview of the four layers and sublayers and how they interact with each other. The first two layers for ZigBee are defined in the IEEE 802.15.4 RF standard as mentioned above.

The interaction of those layers happen through the means of "Service Access Points (SAP)" that offer two services to the upper layer or the application on top of the stack. Those two services are the Data Transmission Service and the Management Service, whereas the Management Service is responsible for administrative tasks only and the Data Transmission Service being responsible for sending or receiving data through the different layers. (Gislason 2008, p. 43)

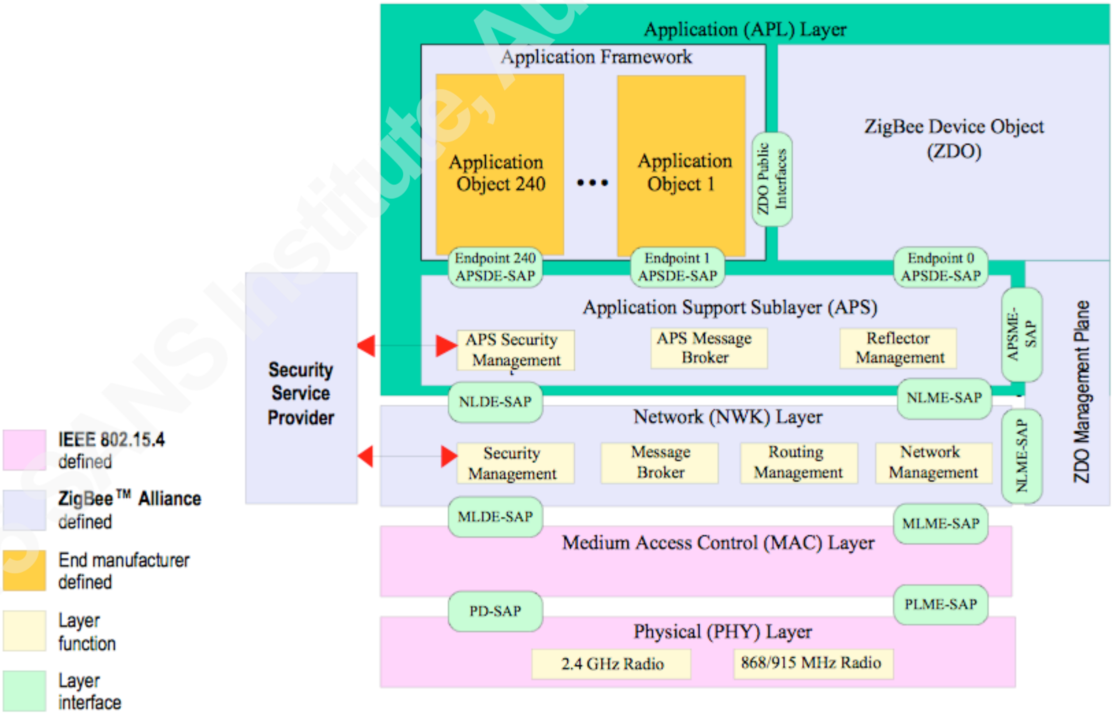


Figure 1. ZigBee Overview (ZigBee Alliance 2008, p.2)

The Network Layer is responsible for managing network formation and routing as well as applying network security and route discovery. (Farahani 2008, p. 109 f)

The Network Layer is also responsible for limiting the number of hops a packet may be forwarded, similar to the Ethernet TTL header field. (Gerstner 2010, p. 10)
Packets are sent in one of the following three communication modes, depending on the requested action (e.g. device discovery is used for unicast or broadcast and multicast is used only for data frames sent to a defined multicast group.). (ZigBee 2008, p. 19)(ZigBee 2008, p. 401f)

- Unicast
- Multicast
- Broadcast

The Application Layer contains three sublayers, which play an important part in security, as will be outlined later in the document.

- Application Support Sublayer (APS)
- ZigBee Device Objects (ZDO)
- Application Framework

The Application Support Sublayer provides driver functionality to ensure correct operations on the ZigBee network by offering a general set of services to the Network and the Application Layer. ZigBee Device Objects (ZDO) are a base class of functionality for interfacing between vendor specific application objects, device profiles and the APS. (ZigBee Alliance 2008, p. 17f). The APS also is responsible for mapping the 64 Bit IEEE addresses to 16 bit ZigBee network addresses and for packet management (ZigBee Alliance 2008, p. 33). Duplicate packets and packets with different profiles as the ones supported by the device are filtered. It also is responsible to maintain binding and grouping tables. (ZigBee Alliance 2008, p. 34ff) (Farahani 2008, p. 122)
The Application framework is responsible for hosting applications that allow further interoperability between the products of different vendors for a specific application. Those application objects are pieces of software developed by the device manufacturer to implement the intended behaviour and feature-set for a device. (Farahani 2008, p. 111). Figure 2 shows how the ZDO, Application Framework and the APS Sublayer interact.

As one device might be a multi-purpose-device, different profiles can be created that define a message format for devices to communicate correctly. Those applications or ZigBee profiles can be used to communication between devices and in case of defined public profiles are used even across vendor boundaries. Profiles are defined by Profile IDs that are defined by the ZigBee Alliance. Public profiles need to use IDs between 0x0001 and 0x7fff and are assigned by the ZigBee Alliance, where vendor specific profiles need to use the IDs between 0xbf00 and 0xffff. (Farahani 2008, p. 22) (Gerstner 2010, p. 19)

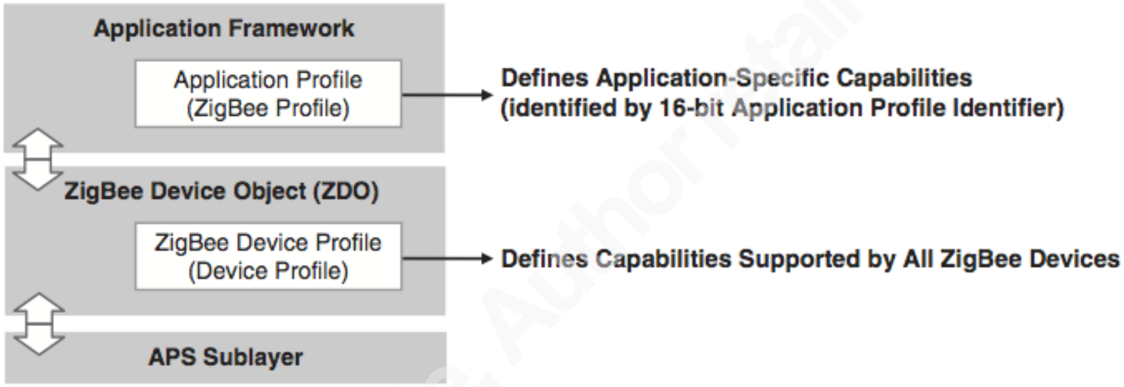

Application Framework
Application Profile (ZigBee Profile)
Defines Application-Specific Capabilities (identified by 16-bit Application Profile Identifier)
ZigBee Device Object (ZDO)
ZigBee Device Profile (Device Profile)
Defines Capabilities Supported by All ZigBee Devices
APS Sublayer

Figure 2. ZDO Interface (Farahani 2008, p.118)

4.1. ZigBee Security

ZigBee includes complex security measures to ensure key establishment, secure networks, key transport and frame security. (ZigBee Alliance 2008, p. 419 f)

Those services are implemented at the Network and the Application Support Sublayer. As a design decision, each ZigBee layer trusts the other layers and every layer is responsible for security, only the Application Support Sublayer provides additional services for devices to maintain a secured relationship to other ZigBee devices. (ZigBee Alliance 2008, p. 423)

Network Layer Security

The ZigBee Network Layer ensures the integrity and encryption of the transmitted frames by applying AES encryption (AES CCM mode) with a key length of 128 bit, and ensures its integrity by using a cipher block chaining message authentication code (CBC-MAC). (ZigBee Alliance 2008, p. 423)

Application Support Layer Security

The Application Layer is responsible for providing the ZDO and installed applications with key establishment and key transport functionality. This layer is also responsible to ensure that packets get encrypted by either the network key or a link key. In the case of network keys, the package is passed down to the Network Layer according to the "each layer" policy. The each layer policy requires that each layer has to take responsibility of initially securing the packets that originate at its layer.(ZigBee Alliance 2008, p. 421)

The APS also sets the security level of the Network Layer and sets active network keys. (ZigBee Alliance 2008, p. 423f)

The ZigBee standard states that the security of ZigBee installations depend on the fact that *"The level of security provided by the ZigBee security architecture depends on the safekeeping of the symmetric keys, on the protection mechanisms employed, and on the proper implementation of the cryptographic mechanisms and associated security policies involved."* (ZigBee Alliance 2008, p. 420). The following conditions need to be met to ensure proper encryption:

- Security protocols such as key establishment need to be implemented and fully executed.
- Secrecy of keys; an unsecured transmission of the key would void the security of the installation.
- Random number generators providing non-predictable random numbers.

Due to resources and cost-per-device issues, ZigBee hardware is most likely not built in a tamper resistant fashion (Goodspeed 2009 p. 1f). The lack of a pre-initialized key may also be due to usability and compatibility reasons. Therefore the initial key that is needed to join a network has to be transmitted in an unsecure way. (ZigBee Alliance 2008, p. 420)

This obviously poses a security risk, and is not limited just because the timeframe when this happens is short. Through jamming, a user can be easily tricked to initiate a factory reset or another way of re-joining, re-establishing that attack time-frame.

An interesting aspect of ZigBee security is based on the fact that there is only one, out of 7 possible (ZigBee Alliance 2008, p. 491) levels of security per network, so devices with different security requirements require different networks to be formed. (ZigBee Alliance 2008, p. 421)

A fundamental part of ZigBee security is the usage and distribution of various keys. Network keys are used in broadcast communication and link keys are used in unicast communication. The link key is only shared between link keys are shared between the two communicating devices and is used only by the Application Support Layer. A link key can possibly be pre-installed at the factory such as ZigBee LightLink devices for example. (E.g. ZLL key) (ZigBee Alliance 2012, 102f).

The network key is shared between all devices on the same network and is either pre-installed or acquired from a Trust Center. For security purposes, ZigBee implements a concept named Trust Center and there is only one Trust Center in each secured network. The Trust Center is a device trusted by other devices within a network that distributes keys. Devices can be pre-loaded with the Trust Center address and master key. (ZigBee Alliance 2008, p. 426).

The Trust Center is able to store multiple keys and defines which one is to be used. The defined key is also called the active key. (ZigBee Alliance 2008, p. 422)
The master key is used to establish a link key with a device, but key installation or handling is the same as with a link key.

A separate transport key for key-exchange is defined to secure the key exchange from a Trust Center and a key-load key is defined to secure the transmission of the master or link keys. (ZigBee Alliance 2008, p. 424). This can happen in a secured fashion for devices already connected to a Trust Center and already communicating through an encrypted channel or in an unsecure fashion to a new device for loading an initial key to the device. In the unsecured mode, the key is not cryptographically protected and should be

communicated and loaded in to the device by using an out-of-band channel. (ZigBee Alliance 2008, p. 425.)
The above mentioned Trust Center is a device responsible for maintaining an internal table of devices, master keys, link keys and network keys in high security mode and only having the network key in a standard security mode.

(Farahani 2008, p. 287). In high security mode, a Trust Center uses secured communications to exchange network keys and possibly unsecured in standard security mode. (ZigBee Alliance 2008, p. 218)

Which security level, which encryption and ultimately which key is used is determined by the security level defined by the application layer. (ZigBee Alliance 2008, p. 490f)
As mentioned above in chapter 4.3 the ZLL key has supposedly been disclosed compromising the security architecture of the ZLL profile.

The following list provides short descriptions of the different key. (ZigBee Alliance 2008, p. 8-11)

- Key-transport key: This is a key derived from a link key used to protect key transport messages carrying a key other than a master key.
- Link key: This is a key that is shared exclusively between two, and only two, peer Application Layer entities within a PAN.
- Master key: This is a shared key used during the execution of a symmetric-key key establishment protocol. The master key is the basis for long-term security between the two devices, and may be used to generate link keys.
- Active network key: This is the key used by a ZigBee device to secure 10 outgoing NWK frames and that is available for use to process incoming NWK 11 frames

4.2. ZigBee Home Automation Profile Security Measures

According to the ZigBee standard, this profile provides standard interfaces and device definitions that allow interoperability among “ZigBee Home Automation devices” that can be used for residential or commercial applications like HVAC, lighting or even door locks. (ZigBee Alliance 2013a, p. 31) All devices need to use a ZigBee Pro stack and support the use of Application link keys. A device can only be certified if it meets certain criteria of this profile, providing the necessary functionality and startup parameters. They are listed in detail in the ZigBee standard, the most interesting part is a hard-coded, pre-defined Trust Center link key and the definition that this key should be used as a fallback solution during startup. The Trust Center address is fixed and set to 0x0000000000000000, the network key is set to NULL as each Trust Center device should generate random keys individually(ZigBee Alliance 2013a, p. 43).

This default fallback Trust Center link key introduces a security risk, according to the Home Automation Profile: *"The current network key shall be transported using the default TC link key in the case where the joining device is unknown or has no specific authorization associated with it. This allows for the case where alternative pre-configured link keys specifically associated with a device can be used as well."* (ZigBee Alliance 2013a, p. 44) As the Trust Center link key is known, the network key can be extracted as soon as it is possible to sniff the initial communication happening during device startup and network join.

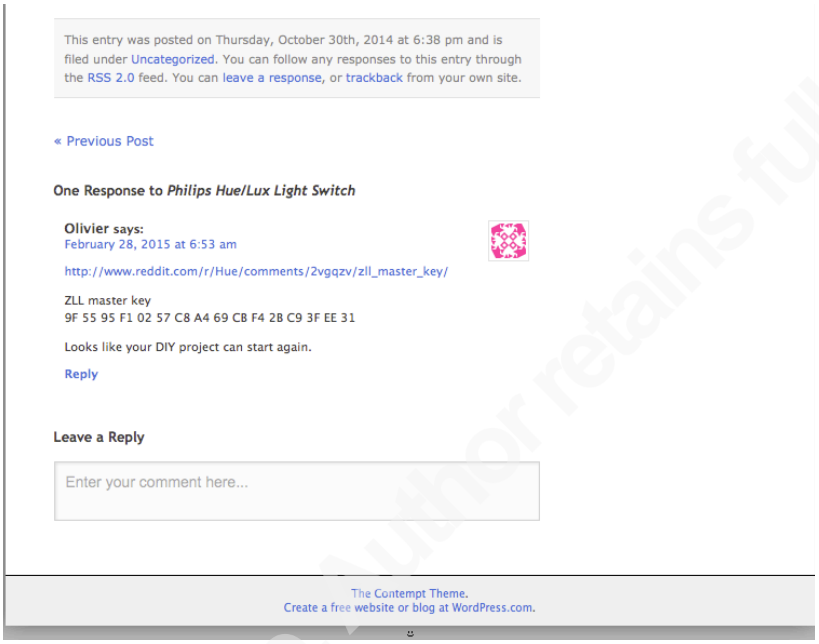
4.3. ZLL

The ZLL (ZigBee Light Link) “profile addresses devices and functionality in the over-the-counter, consumer lighting application domain.”(ZigBee Alliance 2012, p. 1)

ZigBee Light Link uses a pre-defined, non-public key to encrypt the network keys and data communications on network level, as the standard recommends Network Layer security. Each network shall have its own randomly generated network key. (ZigBee Alliance 2012, p. 101)

As the standard states: “The ZLL security architecture is based on using a fixed secret key, known as the ZLL key, which shall be stored in each ZLL device. All ZLL devices use the ZLL key to encrypt/decrypt the exchanged network key. “(ZigBee Alliance 2012, p. 101.)

This is an example of using the common poor method of security called “security by obscurity” and as the ZLL key has supposedly already been disclosed, the security of networks using current ZLL devices has to be considered compromised.



This entry was posted on Thursday, October 30th, 2014 at 6:38 pm and is filed under Uncategorized. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site.

« Previous Post

One Response to *Philips Hue/Lux Light Switch*

Olivier says:
February 28, 2015 at 6:53 am

http://www.reddit.com/r/Hue/comments/2vgqzv/zll_master_key/

ZLL master key
9F 55 95 F1 02 57 C8 A4 69 CB F4 2B C9 3F EE 31

Looks like your DIY project can start again.

Reply

Leave a Reply

Enter your comment here...

The Contempt Theme.
Create a free website or blog at WordPress.com.

Figure 3. Supposedly leaked ZLL key (reddit referenced in a personal blog)

Besides the leaked key, as shown in the screenshot of as blog post in figure 3, ZLL devices support something called a “Touchlink Commissioning” mechanism that allows devices to be paired with controllers.

As the default and publicly known Trust Center link key is used (ZigBee Alliance 2012, p. 81), devices can be “stolen”. (ZigBee Alliance 2012, p.86) Tests showed that with proper radio hardware, a Touchlink Commission is possible from several meters away whereas for security reasons this should only work in close proximity.

The ZLL link key encrypts the network key and is stored in the device (ZigBee Alliance 2012, p.102f.). By knowing the ZLL key it is possible for an attacker to intercept a key-exchange and therefor acquire the current network key. This would then allow the attacker to join the network and control the ZLL devices. This interception can easily be triggered by jamming the target network and waiting for the user to initiate a reset of the devices to restore functionality.

4.4. Other Profiles (Smart Meter)

The Smart Metering Profile is a profile used for energy usage monitoring and distributing the information back to the energy provider. (ZigBee 2013, p. 5) Due to time and device availability constraints, this profile could not be tested.

The Smart Metering Profile is interesting security-wise, as it uses a complex certificate infrastructure and HTTPS requests to ensure encrypted communication that, based on correct certificate checking, should prevent eavesdropping. This is different from other profiles e.g. Home-Automation. The means of providing out-of-band information for registration adds additional security to the device usage. (ZigBee 2013, p. 54-56).

One possible attack here is still the dependency on the secrecy of the pre-loaded keys and the dependency on tamper-resistant hardware as well as an attack or theft of the certificates implemented into the devices and used as either CA certificates or client certificates. (Constantin 2012) (ZigBee 2013, p. 56-68).

4.5. Smart Things Assessment

Two SmartThings Hubs were used as test devices for a practical assessment as they are relatively cheap and were readily available to the author, the second one using a new firmware after the first testing device ceased to function.

Technical specs of the devices tested: (acc. to graph.api.smartthings.com)

1
Model: STH-ETH-001
FW Version: 000.010.00246

2
Model: STH-ETH-001
FW Version: 000.013.00013

The devices need Internet access to send commands to the smartthings.com systems in order to ensure a possible connection from a mobile device to the smartthings.com cloud and back to your SmartThings hub.

Sniffing the traffic sent by the hub indicated the usage of an SSL encrypted connection to a host dc.connect.smartthings.com. See Figure 4 for details.

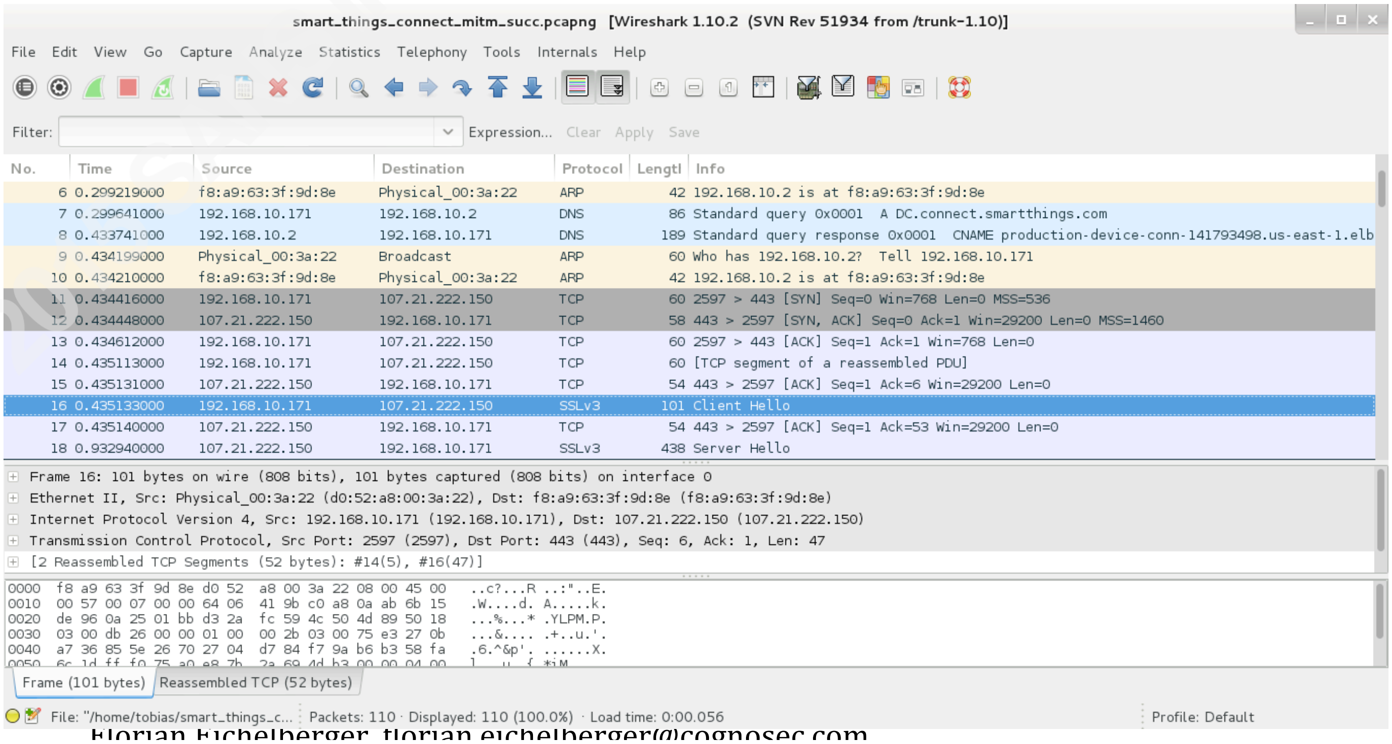
In order to test the validity of the certificate and to see if a man-in-the-middle attack was possible, the following setup was created for testing purposes. As this paper is not primarily on man-in-the-middle attacks, only the relevant information is provided; it is assumed that the reader is able to download and install software or provide easy configurations to standard software.

At first the installation of the used mitmproxy software from http://www.mitmproxy.org (the current release at the time of writing is 0.11.3) is necessary.

To enable the required port forwarding to the proxy software(mitmproxy) , setup of the iptables port forwarding rules is necessary after IP forwarding was activated and NAT masquerading was activated.

To ensure control over the name resolution and the network information provided to the hub via DHCP, a DNS forwarder and a UDHCPD server was setup. By executing the mitmproxy tool using the port that was used before in the iptables forwarding rule, the software listens for incoming connections to intercept and forward.

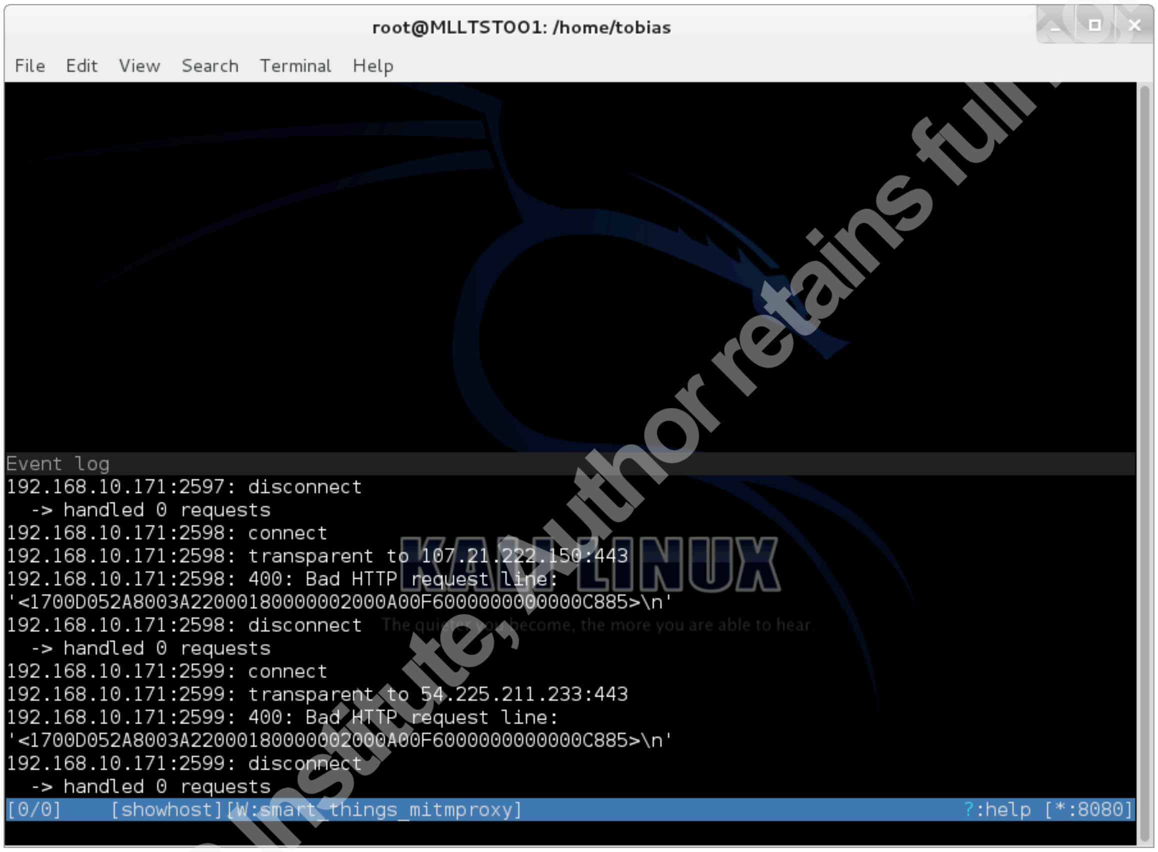
```
mitmproxy -T -port 8080 --host —e
```



```
No.  Time         Source             Destination        Protocol Lengtl Info
   6 0.299219000  f8:a9:63:3f:9d:8e  Physical_00:3a:22  ARP        42 192.168.10.2 is at f8:a9:63:3f:9d:8e
   7 0.299641000  192.168.10.171     192.168.10.2       DNS        86 Standard query 0x0001  A DC.connect.smartthings.com
   8 0.433741000  192.168.10.2       192.168.10.171     DNS       189 Standard query response 0x0001  CNAME production-device-conn-141793498.us-east-1.elb
   9 0.434199000  Physical_00:3a:22  Broadcast          ARP        60 Who has 192.168.10.2?  Tell 192.168.10.171
  10 0.434210000  f8:a9:63:3f:9d:8e  Physical_00:3a:22  ARP        42 192.168.10.2 is at f8:a9:63:3f:9d:8e
  11 0.434416000  192.168.10.171     107.21.222.150     TCP        60 2597 > 443 [SYN] Seq=0 Win=768 Len=0 MSS=536
  12 0.434448000  107.21.222.150     192.168.10.171     TCP        58 443 > 2597 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460
  13 0.434612000  192.168.10.171     107.21.222.150     TCP        60 2597 > 443 [ACK] Seq=1 Ack=1 Win=768 Len=0
  14 0.435113000  192.168.10.171     107.21.222.150     TCP        60 [TCP segment of a reassembled PDU]
  15 0.435131000  107.21.222.150     192.168.10.171     TCP        54 443 > 2597 [ACK] Seq=1 Ack=6 Win=29200 Len=0
  16 0.435133000  192.168.10.171     107.21.222.150     SSLv3     101 Client Hello
  17 0.435140000  107.21.222.150     192.168.10.171     TCP        54 443 > 2597 [ACK] Seq=1 Ack=53 Win=29200 Len=0
  18 0.932940000  107.21.222.150     192.168.10.171     SSLv3     438 Server Hello

⊞ Frame 16: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface 0
⊞ Ethernet II, Src: Physical_00:3a:22 (d0:52:a8:00:3a:22), Dst: f8:a9:63:3f:9d:8e (f8:a9:63:3f:9d:8e)
⊞ Internet Protocol Version 4, Src: 192.168.10.171 (192.168.10.171), Dst: 107.21.222.150 (107.21.222.150)
⊞ Transmission Control Protocol, Src Port: 2597 (2597), Dst Port: 443 (443), Seq: 6, Ack: 1, Len: 47
⊞ [2 Reassembled TCP Segments (52 bytes): #14(5), #16(47)]

0000  f8 a9 63 3f 9d 8e d0 52  a8 00 3a 22 08 00 45 00   ..c?...R ..:"..E.
0010  00 57 00 07 00 00 64 06  41 9b c0 a8 0a ab 6b 15   .W....d. A.....k.
0020  de 96 0a 25 01 bb d3 2a  fc 59 4c 50 4d 89 50 18   ...%...* .YLPM.P.
0030  03 00 db 26 00 00 01 00  00 2b 03 00 75 e3 27 0b   ...&.... .+..u.'.
0040  a7 36 85 5e 26 70 27 04  d7 84 f7 9a b6 b3 58 fa   .6.^&p'. ......X.
0050  6c 1d ff f0 75 a0 e8 7b  2a 69 4d b3 00 00 04 00   l...u..{ *iM

File: "/home/tobias/smart_things_c... Packets: 110 · Displayed: 110 (100.0%) · Load time: 0:00.056
```

Figure 4. Smartthing hub connecting over ssl to it's cloud server, an Amazon WS Server.

See Figure 5 for the intercepted connection and the clear text data transmitted to the smartthings.com server in the eventlog of the tool as it is not interpreted as a correct http connection. It seems to be a proprietary protocol that should be analyzed in more detail.


root@MLLTST001: /home/tobias
File Edit View Search Terminal Help
KALI LINUX
The quieter you become, the more you are able to hear

```
Event log
192.168.10.171:2597: disconnect
  -> handled 0 requests
192.168.10.171:2598: connect
192.168.10.171:2598: transparent to 107.21.222.150:443
192.168.10.171:2598: 400: Bad HTTP request line:
'<1700D052A8003A22000180000002000A00F6000000000000C885>\n'
192.168.10.171:2598: disconnect
  -> handled 0 requests
192.168.10.171:2599: connect
192.168.10.171:2599: transparent to 54.225.211.233:443
192.168.10.171:2599: 400: Bad HTTP request line:
'<1700D052A8003A22000180000002000A00F6000000000000C885>\n'
192.168.10.171:2599: disconnect
  -> handled 0 requests
[0/0]    [showhost][W:smart_things_mitmproxy]                    ?:help [*:8080]
```

Figure 5. The event log of mitmproxy showing the clear text communication.

Even though this is not the full communication, it was sufficient to verify that the man-in-the-middle attack was successful. The attack could be extended by using more complex solution as for example sslsplit from https://www.roe.ch/SSLsplit

In the new recent version of the hub, this issue has been fixed and the certificate is validated properly.

This leads to the conclusion that employing SSL encryption without certificate checking leaves eavesdropping possibilities and is, therefore, a security risk.

5. SDR

Through the availability of powerful and cheap radio hardware, the possibility of a wider range of attacks (e.g. breaking into cars (Greenberg 2014)), eavesdropping police and emergency service communications (Duan 2013) or jamming signals e.g. GPS (Bark 2013) on wireless protocols was made easier and affordable for a broader range of people. The hardware used for the assessments was the USRP (Universal Software Radio Peripheral) device by Ettus Research. The Ettus B210 board covers a broad frequency range from 70 MHz to 6 GHz and connects to the interfacing computer using USB3. It features two channels for receiving and transmitting ("Ettus Research" 2015).

The Ettus B210 is supported by the scapy-radio ("Scapy Radio" 2014) and the GNU Radio framework.("Ettus Research" 2015a)

The board is used in combination with a Lenovo G500, 8 GB notebook because the USB3 chipset (Intel 7 series / C210 xHCI) used seems to be relatively stable in combination with the Ettus board in contrast to the initially used MacBook Pro that could not be used due to stability issues.

6. Scapy-Radio

Scapy is an interactive packet manipulation framework that is able to sniff, decode and send packets for a number of protocols. It can be used for probing, attacks, sniffing, scanning, et cetera. The fact that scapy is more like a framework than a single tool and is open-source allows for much greater flexibility in what it can be used for ("Scapy 2015"). By providing full access to the packet generation and decoding as well as content, it allows for a full range of test cases. To be able to use the Ettus SDR hardware together with scapy, a full working installation of GNU Radio is required and a modified version of scapy has to be used that supports the following protocols (based on GNU Radio 3.7)

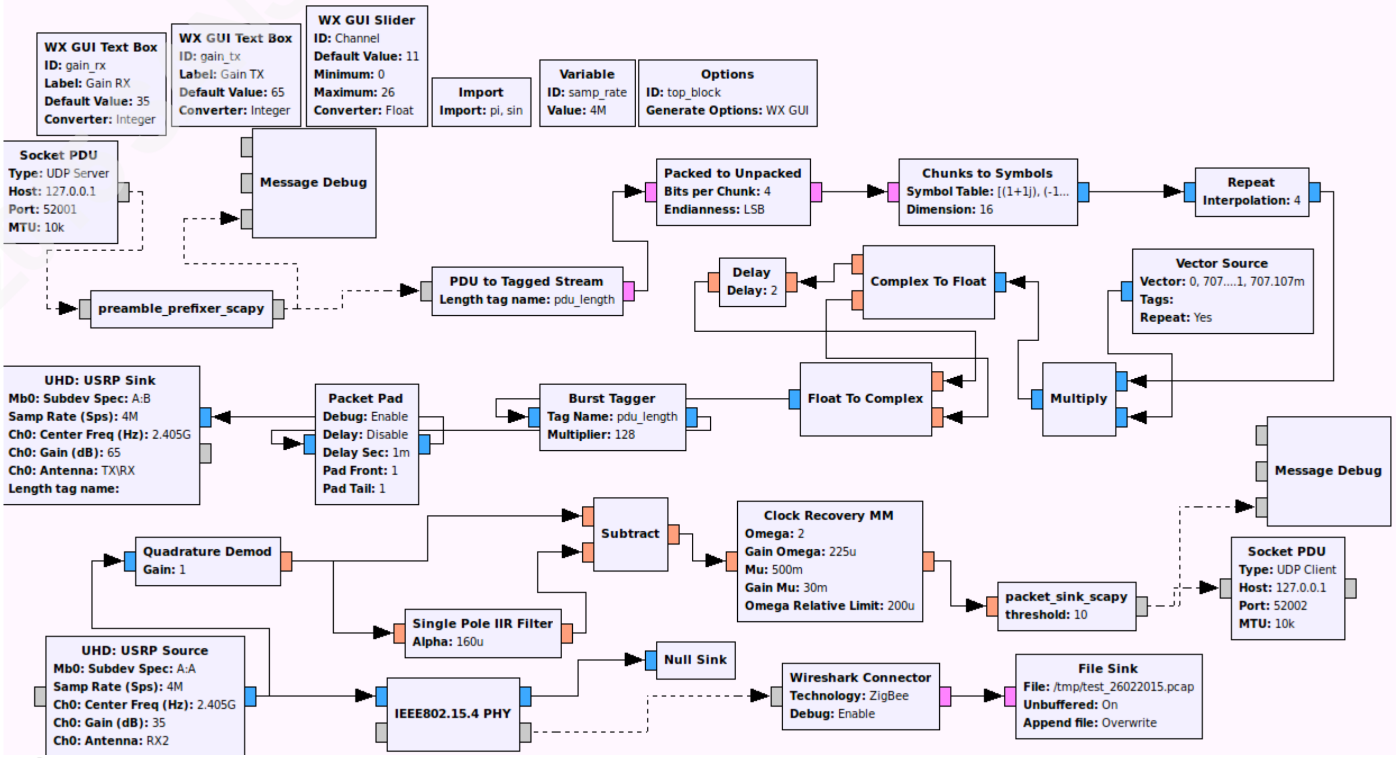
- Bluetooth LE
- 802.15.4 (used by ZigBee, Xbee, 6LoWPAN)

- ZWave (on 868 Mhz)

6.1. GnuRadio companion

Another requirement that must be fulfilled in order to be able to send / receive ZigBee packets is a GNU Radio companion file (.grc). This needs to be created beforehand, in order to “software define” the radio characteristics according to the ZigBee specifications. The GNU Radio companion (“GNU Radio Companion” 2014) is part of the GNU Radio package that is used as a foundation for signal processing tasks in most software defined radio setups. The GNU Radio software version used is 3.7.2.1. The GNU Radio companion provides a graphical interface for signal flow graph creation and generates python code based on those flowcharts that are saved in their own xml format. The GRC files are then compiled to python code that needs to be copied to the location where scapy-radio expects this code to be. This is described in more detail in the chapter 6.3.

GRC files include hierarchical blocks to process the signal that is to be sent or received. See Figure 6 for the used GRC blocks.



WX GUI Text Box
ID: gain_rx
Label: Gain RX
Default Value: 35
Converter: Integer

WX GUI Text Box
ID: gain_tx
Label: Gain TX
Default Value: 65
Converter: Integer

WX GUI Slider
ID: Channel
Default Value: 11
Minimum: 0
Maximum: 26
Converter: Float

Import
Import: pi, sin

Variable
ID: samp_rate
Value: 4M

Options
ID: top_block
Generate Options: WX GUI

Socket PDU
Type: UDP Server
Host: 127.0.0.1
Port: 52001
MTU: 10k

Message Debug

preamble_prefixer_scapy

PDU to Tagged Stream
Length tag name: pdu_length

Packed to Unpacked
Bits per Chunk: 4
Endianness: LSB

Chunks to Symbols
Symbol Table: [(1+1j), (-1...
Dimension: 16

Repeat
Interpolation: 4

Delay
Delay: 2

Complex To Float

Vector Source
Vector: 0, 707....1, 707.107m
Tags:
Repeat: Yes

UHD: USRP Sink
Mb0: Subdev Spec: A:B
Samp Rate (Sps): 4M
Ch0: Center Freq (Hz): 2.405G
Ch0: Gain (dB): 65
Ch0: Antenna: TX\RX
Length tag name:

Packet Pad
Debug: Enable
Delay: Disable
Delay Sec: 1m
Pad Front: 1
Pad Tail: 1

Burst Tagger
Tag Name: pdu_length
Multiplier: 128

Float To Complex

Multiply

Message Debug

Subtract

Clock Recovery MM
Omega: 2
Gain Omega: 225u
Mu: 500m
Gain Mu: 30m
Omega Relative Limit: 200u

Quadrature Demod
Gain: 1

packet_sink_scapy
threshold: 10

Socket PDU
Type: UDP Client
Host: 127.0.0.1
Port: 52002
MTU: 10k

Single Pole IIR Filter
Alpha: 160u

UHD: USRP Source
Mb0: Subdev Spec: A:A
Samp Rate (Sps): 4M
Ch0: Center Freq (Hz): 2.405G
Ch0: Gain (dB): 35
Ch0: Antenna: RX2

Null Sink

IEEE802.15.4 PHY

Wireshark Connector
Technology: ZigBee
Debug: Enable

File Sink
File: /tmp/test_26022015.pcap
Unbuffered: On
Append file: Overwrite

There are three separate parts of the shown workflow.
The top row of blocks represent the options to be setup by the user and to be used by other blocks, it is important to set the correct gain levels to match the distance between the SDR (USRP) hardware and the device to attack. As the gain on the sending and receiving side is quite different, it was necessary to provide two independent values. The gain value is important as a too-low gain results in packets not being received by the target and a too-high gain on the receiving side will result in packets being incorrectly received and too much noise being received. The sample rate/channel needs to be set correctly in order to be able to send packets.

The following row, starting with the “Socket PDU” block and by following the arrows ending with the UHD USRP Sink block is the part of the workflow responsible for generating and transmitting the signal. The receiving part of the workflow is shown, starting with the “UHD USRP Source” block and then demodulating and reconstructing the data from the signal. This data is then provided to scapy-radio and written to a pcap file through the Wireshark Connector.

Those GRC blocks used in this paper heavily rely on the implementation that was provided by Thomas Schmid (Schmid 2006) from UCLA and was later extended by a team of the University of Innsbruck, Austria (Bloessl, Leitner, Dressler & Sommer 2013).

6.2. Encryption

In order to allow command injection for ZigBee, encryption support needs to be available to the scapy-radio software. Scapy-radio originally did not support ZigBee encryption, as ZigBee only supports the use of AES encryption in CCM* mode 128-bit block size, requires one key and supports message authentication and encryption or only message encryption. (ZigBee Alliance 2008, p. 521)

In order to leverage the ZigBee encryption in scapy-radio, the encryption capabilities that were already included into the killerbee software, a set of tools for ZigBee attacks (Wright, Melgares 2009) have been ported to a recent scapy-radio version (version 2015-02-20). The code that has been adapted and ported to scapy-radio was taken from a recent killerbee version (rmspeers 2014) .The code is available on request from the author, an attack example is provided in chapter 6.3

6.3. Command Injection

As an attack that uses the encryption implemented, the author chose to demonstrate ZigBee command injection to inject ZigBee commands to turn a light off and on. A test setup was created to sniff packets sent by the deCONZ RaspBee shield ("RaspBee" 2015) to the lightbulb. The packets are intercepted with the B210 SDR board, decrypted, the content manipulated, re-encrypted and send again with a changed command.

The lightbulb used in this example was a Philips Hue LWB004 with a Datecode of 20140324.

Initially the radio hardware had to be setup. To be able to use the radio hardware, the device was plugged into a USB 3 port for data throughput reasons and the following commands executed after having installed the USRP Hardware Driver (UHD) from Ettus research

```
./uhd_usrp_probe
```

By using the following command, the device was identified and verified as working:

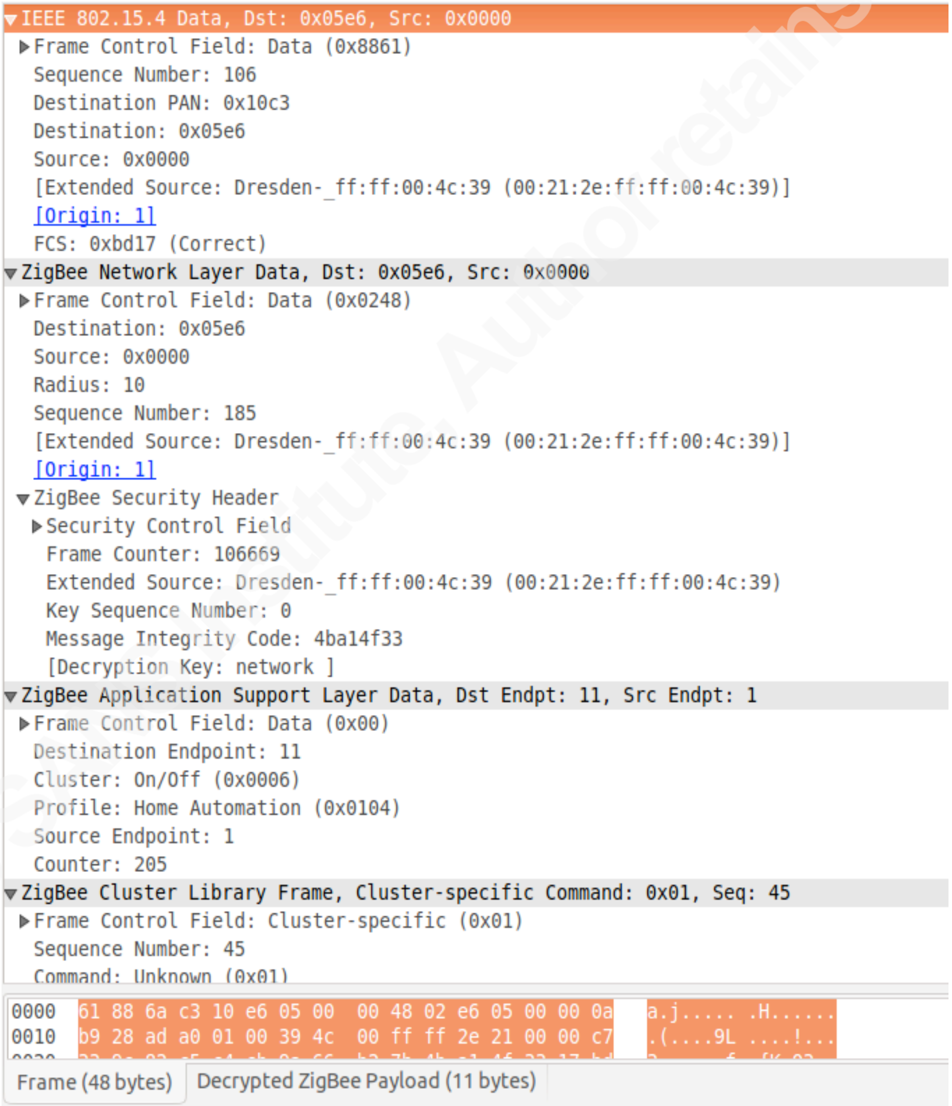
```
./uhd_find_devices
```

After verification that the device was working, a python script written for this attack was executed that uses scapy-radio with the newly implemented encryption and the imported killerbee module to manipulate and send the commands injected. In order for the python script to be able to access the SDR hardware via scapy, the python file generate from the flowchart as described in chapter 6.1 is required. It should be noted that scapy expects to see the compiled flowchart script in the following directory:

```
$HOME/.scapy/radio/
```

The demo script manipulating and sending the packets is available on request from the author. By executing scapy-radio commands in the demo script, the python file compiled in GNU Radio gets executed and a packet handling the callback function gets called for every packet that is received. A sniffed packet originating from deCONZ RaspBee used to send the “on” command to the light is shown in Figure 7.

As seen on Figure 7, there are several header fields that need to be correctly set in order to be able to inject a command as those fields need to be correctly increased as they provide freshness checks and replay protection for the ZigBee communications. Following is a list of the header fields that are adjusted, the fields within the APS header and payload or the ZCL cluster header and payload are encrypted by the current network key and can only be changed after decryption. For our example the network key was



```
▼IEEE 802.15.4 Data, Dst: 0x05e6, Src: 0x0000
  ▶Frame Control Field: Data (0x8861)
   Sequence Number: 106
   Destination PAN: 0x10c3
   Destination: 0x05e6
   Source: 0x0000
   [Extended Source: Dresden-_ff:ff:00:4c:39 (00:21:2e:ff:ff:00:4c:39)]
   [Origin: 1]
   FCS: 0xbd17 (Correct)
▼ZigBee Network Layer Data, Dst: 0x05e6, Src: 0x0000
  ▶Frame Control Field: Data (0x0248)
   Destination: 0x05e6
   Source: 0x0000
   Radius: 10
   Sequence Number: 185
   [Extended Source: Dresden-_ff:ff:00:4c:39 (00:21:2e:ff:ff:00:4c:39)]
   [Origin: 1]
  ▼ZigBee Security Header
    ▶Security Control Field
     Frame Counter: 106669
     Extended Source: Dresden-_ff:ff:00:4c:39 (00:21:2e:ff:ff:00:4c:39)
     Key Sequence Number: 0
     Message Integrity Code: 4ba14f33
     [Decryption Key: network ]
▼ZigBee Application Support Layer Data, Dst Endpt: 11, Src Endpt: 1
  ▶Frame Control Field: Data (0x00)
   Destination Endpoint: 11
   Cluster: On/Off (0x0006)
   Profile: Home Automation (0x0104)
   Source Endpoint: 1
   Counter: 205
▼ZigBee Cluster Library Frame, Cluster-specific Command: 0x01, Seq: 45
  ▶Frame Control Field: Cluster-specific (0x01)
   Sequence Number: 45
   Command: Unknown (0x01)

0000  61 88 6a c3 10 e6 05 00  00 48 02 e6 05 00 00 0a   a.j..... .H......
0010  b9 28 ad a0 01 00 39 4c  00 ff ff 2e 21 00 00 c7   .(....9L ....!...
```

Figure 7. The "on" packet sent to the lightbulb.

known, but the network key could be obtained by either sniffing the key-exchange as described in the Standard (ZigBee Alliance 2008, p. 449-453) (ZigBee Alliance 2008, p. 475-479) or by extracting the key from a device.

The APS and ZCL header and payload frame format can be found in the ZigBee Standard for further reference. (ZigBee Alliance 2008 , p. 52-59)

- IEEE Header
  - Sequence Number
  - Increased with every packet sent. It specifies the sequence identifier for the frame. (IEEE 2011, p. 59).
- ZigBee Network Layer
  - Sequence Number
  - Used to uniquely identify a frame in combination with the source address. (ZigBee Alliance 2008, p. 382)
- ZigBee Security Header (Auxiliary header)
  - Frame Counter is used to make sure no duplicate packets are processed. (ZigBee Alliance 2008, p. 491)

From here the data is encrypted in the packet (ZigBee Alliance 2008, p.

- ZigBee Application Support Layer (APS)
  - Counter
  - This counter is used to detect duplicate frames and ensure that those frames are only processed once. (ZigBee Alliance 2008, p. 67)
- ZigBee Cluster Library (ZCL)
  - Sequence
  - This sequence number is increased for every transmission of a ZCL command. (ZigBee Alliance 2013, p. 85)

After having acquired the network key and by sniffing ZigBee packets sent to a potential target it is possible to record the currently used sequence numbers and frame counters after decryption. Initially the tests were conducted by incrementing the fields mentioned above and setting the correct functions and parameters in the headers. Later tests showed that the ZigBee Security Header Frame Counter and the IEEE header Sequence Number are the only fields that need to be incremented to inject commands, in our case an “on” or “off” command switching the light accordingly. This is not only possible at close range, instead it has been tested to be working for distances of 2,5 – 3 meters at least. This distance is sufficient enough to sniff packets from an unsuspecting location and sending commands even when e.g. outside a building to devices in the building.

Our test also showed that sending the same command packet several times resulted in the packets being acknowledged at the IEEE layer, indicating that the IEEE Sequence number is not used as a replay protection.

7. Conclusion

As the initial assessments listed in chapter 4.5 showed, the usage of SSL / TLS encryption is getting more and more widespread yet the usage of proper certificate validation to prevent man-in-the middle attacks is missing in many devices. By using pre-defined and published keys for key transport encryption, a secure data transmission is no longer possible and relying on the secrecy of keys distributed only among a limited group of people is a security method known to have failed before. (e.g. “Content Scramble System” 2015). Travis Goodspeed showed successful attacks on ZigBee hardware to extract keys (Goodspeed 2009 p. 1f), and thus without appropriate hardware key secrecy should not be the foundation of a product’s security architecture.
Through the availability of inexpensive and powerful radio hardware, attacks through sniffing and command injection are most likely becoming more widespread. The proposed software allows someone to sniff data but also allows that person to perform command injection commands. The replay protection that is built into the ZigBee protocol is working as it was shown it is not possible to replay a packet, however command injection is still possible.

8. Future Work / Limitations

A larger extension of the scapy-radio framework would be a rewarding target to implement more attack scenarios. The creation and extensions of frameworks for other home-automation protocols should be encouraged to allow researches to find critical security holes before a massive widespread usage of various “Smart Things” takes place. If design or implementation flaws are discovered, replacing these devices once they have been deployed would be very costly.

The framework could be extended to allow automatic detection of devices, key extraction if public keys are used or automatic jamming in order to trick the user into re-pairing the devices so the key-exchange can be sniffed before injecting commands.

9. Bibliography

Ashton, K., (2009, June 22) That ‘Internet of Things’ Thing. Retrieved from http://www.rfidjournal.com/articles/view?4986

Bark S.,(2013, January 22). Mobile Hack Attacks to Increase with SDR Adoption. Retrieved from https://www.digitalassurance.com/about-us/press/mobile-hack-attacks-increase-sdr-adoption

Bennet B., (2009, August 23). What happened to the Internet fridge ?. Retrieved from http://billbennett.co.nz/2009/08/23/happened-internetconnected-fridge/

Bloessl B., Leitner C., Dressler F., & Sommer F., (2013). A GNU Radio-based IEEE 802.15.4 Testbed. Proceedings of 12. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN 2013), 37-40.

Content Scramble System. (2015). Retrieved from http://en.wikipedia.org/wiki/Content_Scramble_System

Dalrymple S.D., (2014). Comparison of Zigbee Replay attacks using a universal software radio peripheral and usb radio. Master Thesis. Retrieved from http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA601406

Duan S., (2013, June). Security Analysis of TETRA. Master's thesis. Retrieved from http://www.diva-portal.org/smash/get/diva2:656471/FULLTEXT01.pdf

Ettus Research (2015). Retrieved from http://www.ettus.com/product/details/UB210-KIT

Ettus Research (2015a). Retrieved from http://www.ettus.com/kb/detail/software-defined-radio-usrp-and-gnu-radio-tutorial-set

Evans-Pughe, C. (2003). Bzzzz zzz [ZigBee wireless standard]. IEE Review, 49(3), 28-31.

Farahani, S. (2008). ZigBee wireless networks and transceivers. Amsterdam: Newnes/Elsevier.

Gartner, (2014a). Gartner Identifies the Top 10 Strategic Technology Trends for 2015, Retrieved from http://www.gartner.com/newsroom/id/2867917

Gartner, (2014b). Gartner Says a Typical Family Home Could Contain More Than 500 Smart Devices by 2022. Retrieved from http://www.gartner.com/newsroom/id/2839717

Gerstner, M. (2009). ZigBee. Bachelor Thesis Paper. Retrieved from https://www.auto.tuwien.ac.at/bib/pdf_TR/TR0148.pdf

Gislason, D., (2008). Chapter 1 - Hello ZigBee. Burlington: Newnes. Retrieved from http://www.sciencedirect.com/science/article/pii/B978075068597900001X

Goodspeed, T. (2009), Extracting Keys from Second Generation Zigbee Chips. Black Hat USA, Las Vegas.

Greenberg A., (2014, April 08). Watch this wireless hack pop a car's locks in minutes. Retrieved from http://www.wired.com/2014/08/wireless-car-hack/

GNU Radio Companion. (2014). Retrieved from https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion

Kennedy, J. (1926, January 26). When woman is Boss. Collier's Illustrated Weekly 77, 163-180.

IEEE, (2011, September 05). IEEE Standard for Local and metropolitan area networks—

Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) New York, United States. IEEE Doccument STDPD97126.

Laird, D., & Laird, E. (1964). How to get along with automation. New York: McGraw-Hill.

Lutolf, R., (1992). Smart Home concept and the integration of energy meters into a home based system. Proceedings of the Seventh International Conference on Metering Apparatus and Tariffs for Electricity Supply. 277-278.

Markert J., Massoth M., Fischer K.P., Furnell S.M., & Bolan C., (2011). Attack Vectors to Wireless ZigBee Network Communications - Analysis and Countermeasures. Proceedings of the Seventh Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN 2011). 57-66.

McLuhan, M. (1966). Understanding Media: The Extensions of Man.

Melgares R. A., (2011). 802.15.4/ZigBee Analysis and Security: Tools for Practical explorations of the attack surface. Dartmouth Computer Science Technical Report TR2011-689.

Miorandi, D., Sicari, S., Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. Ad Hoc Networks, 10(7), 1497-1516.

Obaid. T.,Rashed H., Abou-Elnour A., Rehan M., Saleh M. & Tarique M., (2014). ZIGBEE TECHNOLOGY AND ITS APPLICATION IN WIRELESS HOME AUTOMATION SYSTEMS: A SURVEY. International Journal of Computer Networks & Communications (IJCNC) 6(4), 115-131.

Payne T., (2015, March 25). The IoT’s stalled start. Retrieved from http://www.itp.net/602584-the-iots-stalled-start

Poole, I. What exactly is … ZigBee?, (2004). Communications Engineer 2(4), 44-45.

Rathnayaka A. J. D., Potdar, V. M., Kuruppu, S. J., (2011). Evaluation of Wireless Home Automation Technologies. 5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011), 76-81.

rmspeers (2014, November 28). scapy_extensions.py. Retrieved from https://github.com/riverloopsec/killerbee/tree/edf740163b8f4e8ac14d295730c0c931f0f6cceb/killerbee/killerbee/scapy_extensions.py

RaspBee. (n.d.). Retrieved from https://www.dresden-elektronik.de/funktechnik/products/boards-and-kits/development-boards/raspbee/

Ross F, (1958). Automation:Servant to Man.New York: Lothrop, Lee & Shepard.

Santucci, G., (2010). Vision and Challenges for Realising the Internet of Things. Brussels: Publications Office of the European Union.

Scapy. (2015). Retrieved from http://www.secdev.org/projects/scapy/

Scapy Radio. (2014). 2015, from http://bitbucket.cassidiancybersecurity.com/scapy-radio/

Schmid, T., (2006). GNU Radio 802.15. 4 En-and Decoding" Networked & Embedded Systems Laboratory, UCLA, Technical Report TR-UCLA- NESL-200609-06, June 2006.

Steinbuch K., (1966). Die informierte Gesellschaft. Stuttgart: Deutsche Verlags-Anstalt.

Sundmaeker, H., Guillemin,P.F., and Woelffle, S., (2010). Vision and Challenges for Realising the Internet of Things. Brussels: Publications Office of the European Union.

Wright, J., & Melgares, R. (2009, January 1). Riverloopsec/killerbee. Retrieved from https://github.com/riverloopsec/killerbee/tree/master/killerbee

Xing, K., Sundhar Rajamadam Srinivasan, S., Rivera, M., Li, J., & Cheng, X. (2010). Attacks an Countermeasures in Sensor Networks: A Survey. In Network Security, 251-272.

ZigBee Alliance, (2008). ZIGBEE SPECIFICATION San Ramon, United States. ZigBee Document 053474r17.

ZigBee Alliance (2012). ZigBee Light Link Standard. San Ramon, United States. Version 1.0, ZigBee Document 11-0037-10.

ZigBee Alliance (2013). Smart Energy Profile 2 Application Protocol Standard. San Ramon, United States. Version 2, ZigBee Document 13-0200-00.

ZigBee Alliance, (2013a). ZIGBEE HOME AUTOMATION PUBLIC APPLICATION PROFILE. San Ramon, United States. Revision 29, Version 1.2, ZigBee Document 05-3520-29.

ZigBee Alliance. The Alliance. (2015). Retrieved from http://zigbee.org/zigbeealliance