IoTSSC – The Cloud

What is the Cloud?

"A network of remote servers hosted on the Internet and used to store, manage, and process data in place of local servers or personal computers."

(Oxford Dictionaries)

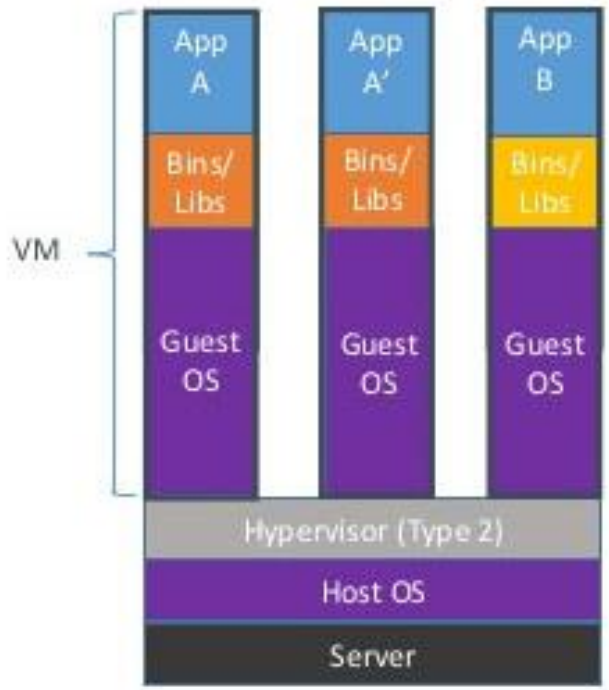
Crucial component of IoT systems

- Device management and configuration
- Data aggregation, processing, storage, and analysis/ visualisation
- Service customisation and infrastructure sharing

Virtualisation

- Virtualisation is what enables clouds to run separate workloads under strict resource partitioning
- Such separation allows to optimally utilise CPU and memory, while providing security guarantees
- Two approaches:
  - Virtual Machines – multiple instances of potentially different OSes running on the same physical machine (Infrastructure as a Service – IaaS)
  - Containers – different applications running on a virtualised OS within partitions. Execution safe to the kernel even if apps may have security issues (Platform as a Service – PaaS)

Virtual Machines vs Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

docker

Advantages of the Container Approach

FAST TO INSTANTIATE

CAN BE DESTROYED AS NEEDED

NO NEED FOR A HYPERVISOR

OS LIBRARIES CAN BE SHARED

EASY TO SCALE

Example (Simple web page - https://docs.docker.com/get-started/)

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt
# Make port 80 available to the world outside this container
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Example (continued)

app.py basically listens for connection on port 80 and returns an HTML page

To run the app, simply call

docker run -p 4000:80 docker-file

-p maps port 80 of the container to port 4000 of the localhost

Putting everything together

We talked about

- How to program embedded devices
- How to connect them to an Internet gateway wirelessly
- How to secure their communication
- How to instantiate apps in the cloud that could do meaningful things with the data IoT devices may generate

Remaining question: how to actually integrate the devices with the cloud?

Simplest way: HTTP

- Set up a lightweight service with HTTP transport and REST (REpresentational State Transfer) API
- IoT device periodically collects measurements and packages them into JSON format

```
record={
        “date”: “2018-04-04”,
        “time”: “09:30:00”,
        “temperature”: 20.1
}
```

- Sends these as HTTP POST requests to the server

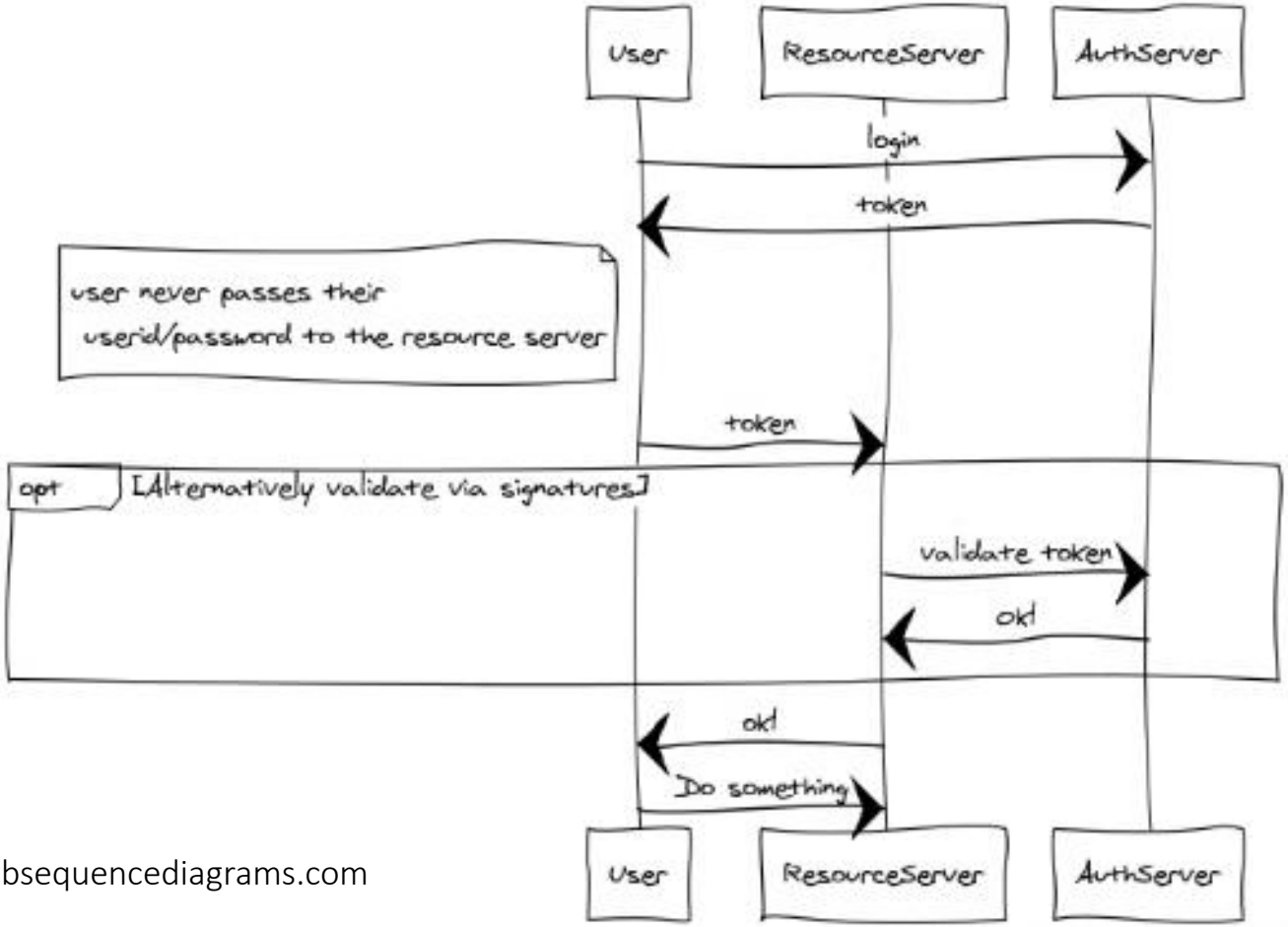
Sending the post request

- You can do this e.g. in Python

```
import requests
import json
…
url=“http://<uri_of_server_end_point”
requests.post(url, data=json.dumps(measurements))
```

How to ensure only authorised devices send data?

Use OAuth (authorisation framework – RFC 6749/50)

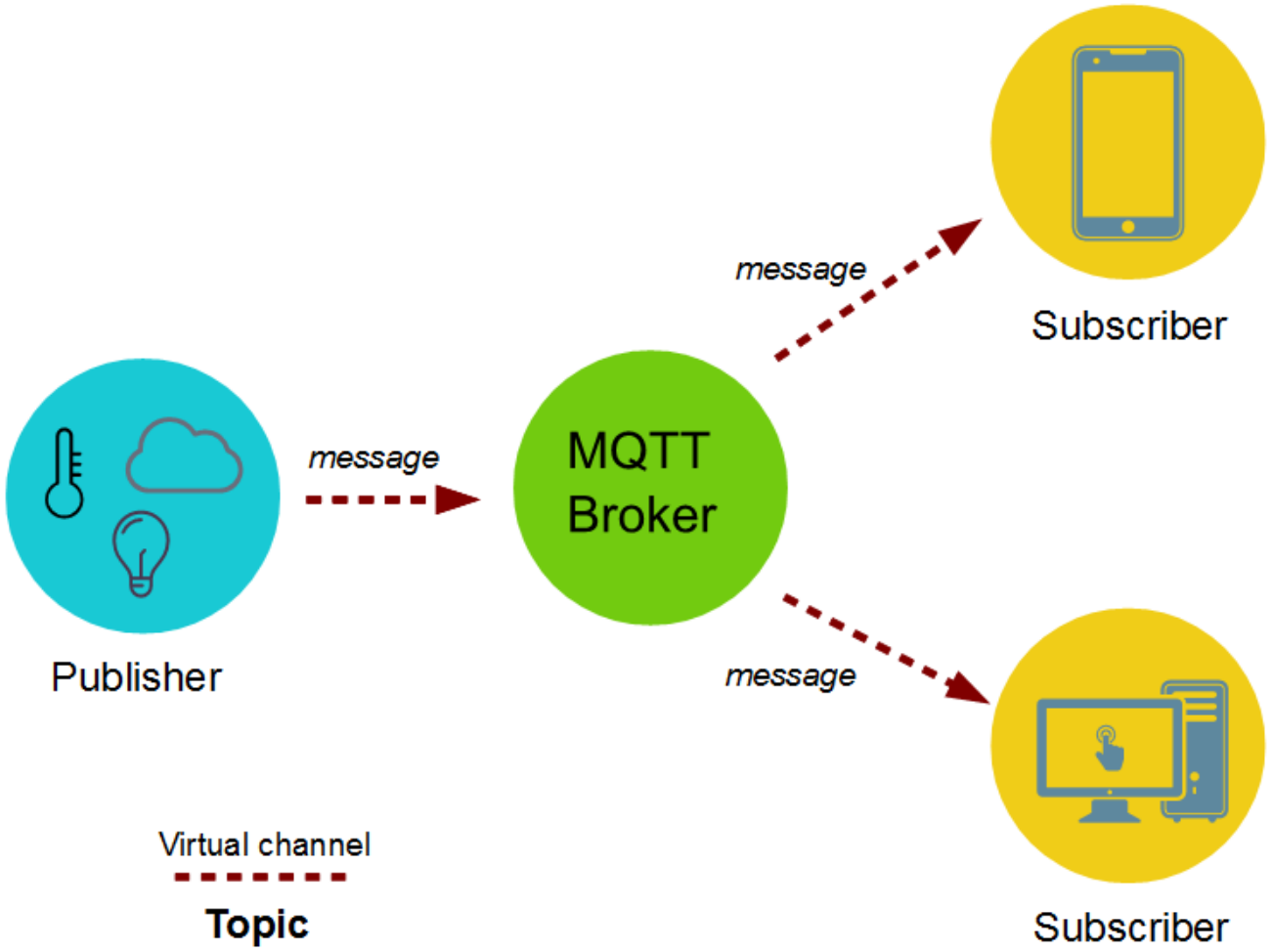


www.websequencediagrams.com

MQ Telemetry Transport (MQTT)

- MQTT is a messaging protocol that allows users to collect information from constrained IoT devices (or devices to communicate among each other – M2M) following a publish/subscribe paradigm
- Networking fabric assumed to be low bandwidth, high latency
- Works at the application layer on top of TCP/IP
- Originally part of IBM's "message queue" suite for industrial applications
- Current architecture centred around a broker/server

MQTT


message
Subscriber
MQTT
Broker
message
Publisher
message
Virtual channel
Topic
Subscriber

The MQTT broker

- Dispatches messages received from publishers (different IoT devices) to the clients (subscriber)
- Different topics used as filters – possible to combine multiple topics into topic levels
- Topics create ‘virtual channels’, which decouple the publishers from the subscribers – scalability
- Three Quality of Service (QoS) levels defined – refer to the guarantees of delivering a message

MQTT QoS

- Important feature – client can choose QoS level based on network reliability
- Three levels defined:
  - QoS 0 – at most once: best-effort delivery (message not acknowledged by the recipient, nor stored and redelivered if subscriber unreachable)
  - QoS 1 – at least once: guaranteed that the messages delivered at least once to the receiver (though may be delivered more than once); message stored by sender until acknowledge by the receiver (PUBACK)
  - QoS 2 – each message received exactly once (safest but slowest)

Retained messages

- A publishing client has no guarantee that a message is received by a subscriber.
- Only guarantees that message is delivered to the broker.
- Likewise, the subscriber has no guarantees about when it will received the first message on a topic, as this depends strictly on publisher.
- Retain messages are like normal MQTT messages but with the ‘retain’ flag set -> broker stores the last message with retain flag, and the QoS for that topic
- Clients receive these messages immediately after subscribing.

Retained messages

- Subscriber receive a message even if not subscribed to an exact topic but to the relevant topic level, e.g.
  - Client A subscribes to /home/#
  - Client B publishes a retained messages to /home/bedroom1/temperature
  - Client A receives the last temperature reading for that room as soon as it subscribes
- To delete a retained message, the publisher sends a new retained message with zero payload.
- On Linux, you can try mosquitto (broker) and mosquitto-clients

Constrained Application Protocol (CoAP) – RFC 7252

Very similar to HTTP, uses URIs (coap://), but

- Works on top of UDP (faster), reliability handed off to retransmissions implemented at Application layer
- Incorporates mechanism for discovering other devices
- Multicast support (send messages to groups of recipients)
- Asynchronous messaging (no need to establish a session)
- Caching (store requests and responses)
- Simple 4-byte packet header (low overhead)

CoAP features

- A node can act both as client and server – no need for a central entity (unlike MQTT that needs a broker)
- Subscriptions – used when devices await updates, or to know if the state of a device has changed
- Firmware updates can be performed with Block Mode transmissions – large files divided into smaller parts
- Additional security layer (DTLS) that can be implemented between UDP and CoAP

IoT Security

Image: xkcd

How is securing IoT different from securing any other system?

- IoT is a system of systems, with
- A great deal of heterogeneity (sensing, computation, communication, energy)
- New types of devices appearing all the time
- Some discontinued (e.g. due to vendor bankruptcy), yet still on the market (e.g. Jawbone)
- Software continuously evolving

Classic pillars of Information Assurance remain the same

- Confidentiality – keeping information secret
- Integrity – ensuring information not modified (accidentally or purposely) without being detected
- Authentication – data originates from a known (identified) end point
- Non-repudiation – the actions of a user/system cannot be denied once performed
- Availability – the system/data is available when needed

Identifying vulnerabilities and mitigating risks

- Start by acquiring in depth knowledge of the architecture and the key entities involved
- Identify potential entry points for each entity
- Understand the data flow paths
- Define trust boundaries (typically the IoT device considered the easiest to compromise, but the trust boundary could be pushed to the GW, if secured)
- Conceive plausible attack scenarios (this is mostly based on previous experience plus curiosity)

Threat modelling

- The process of analysing in a structured way the vulnerabilities of a system from the point of view of a hypothetical attacker
- Helps identifying risks, quantifying their likelihood and potential severity, and prioritising mechanisms for mitigation/prevention
- Different methodologies/taxonomies
  - STRIDE (Microsoft)
  - Open Threat Taxonomy
  - ITU-T X.800
  - ENISA (general – Jan. 2016 + IoT specific – Nov. 2017)
  - OWASP IoT
  - Others

STRIDE

- Strictly speaking not IoT specific, but threats can be applicable across different components of an IoT systems/service

Threat	Relevant IA properties
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation, Auditing
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privileges	Authorisation

Open Threat Taxonomy

- Physical (actions that could lead to system theft/harm destruction – e.g. HVAC failure, natural disaster, vandalism)
- Resource (failure of information system due to disruption of resources required for operation – e.g. power supply, communication services)
- Personnel (deliberate or accidental actions of employees that can harm a system, social engineering)
- Technical (technical actions by an attacker that could harm the information system)

- The CIA triad applies across all these groups.
- Multiple threads within each group, rated from 1 to 5.

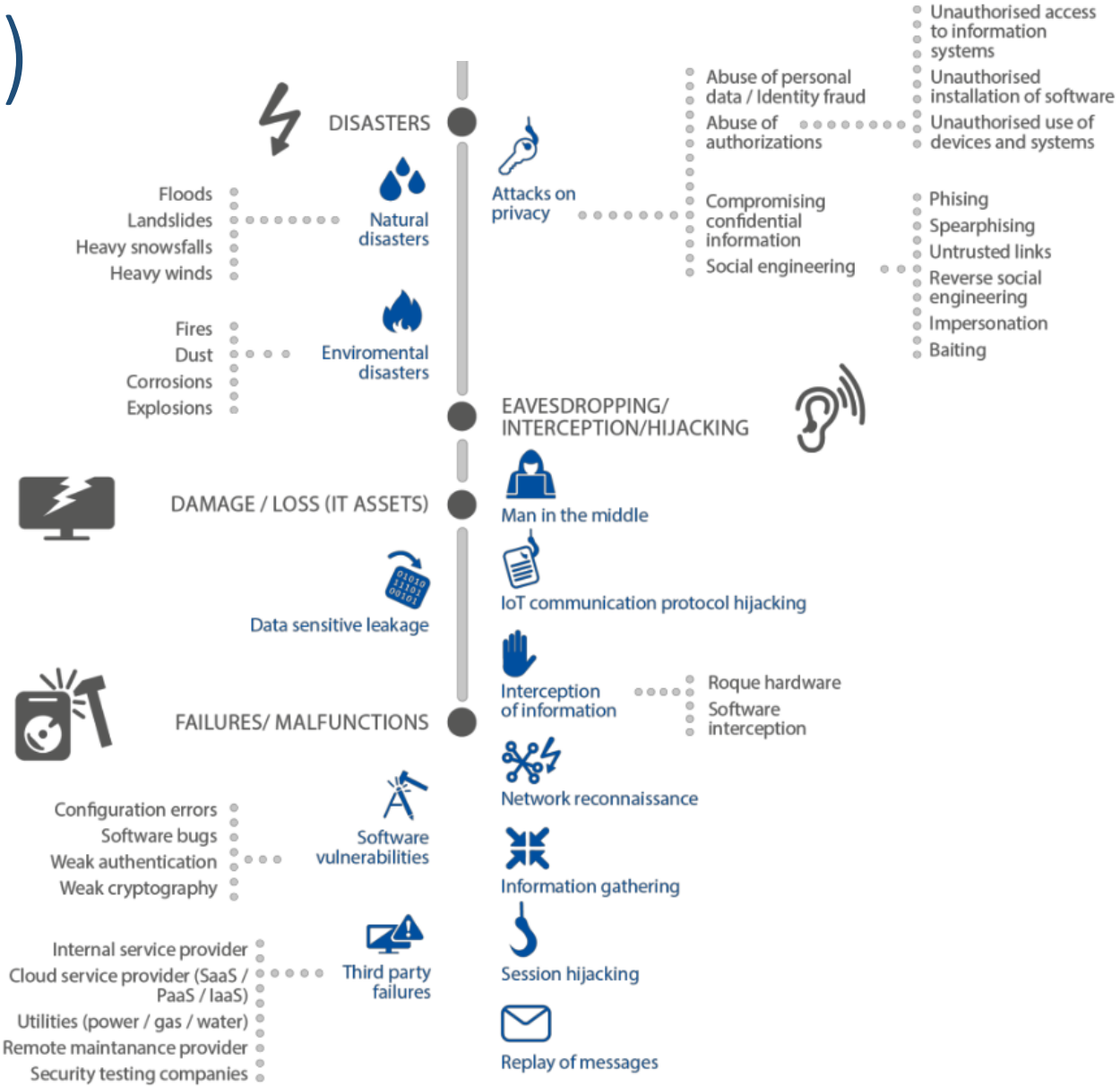
Open Threat Taxonomy – Technical threats

- Credential discovery (sniffing, brute forcing, cracking)
- Escalation/abuse of system privileges
- Cryptanalysis
- DoS
- Data capture/manipulation in transit
- App exploitation via code injection, reverse engineering, input manipulation, API abuse, etc.

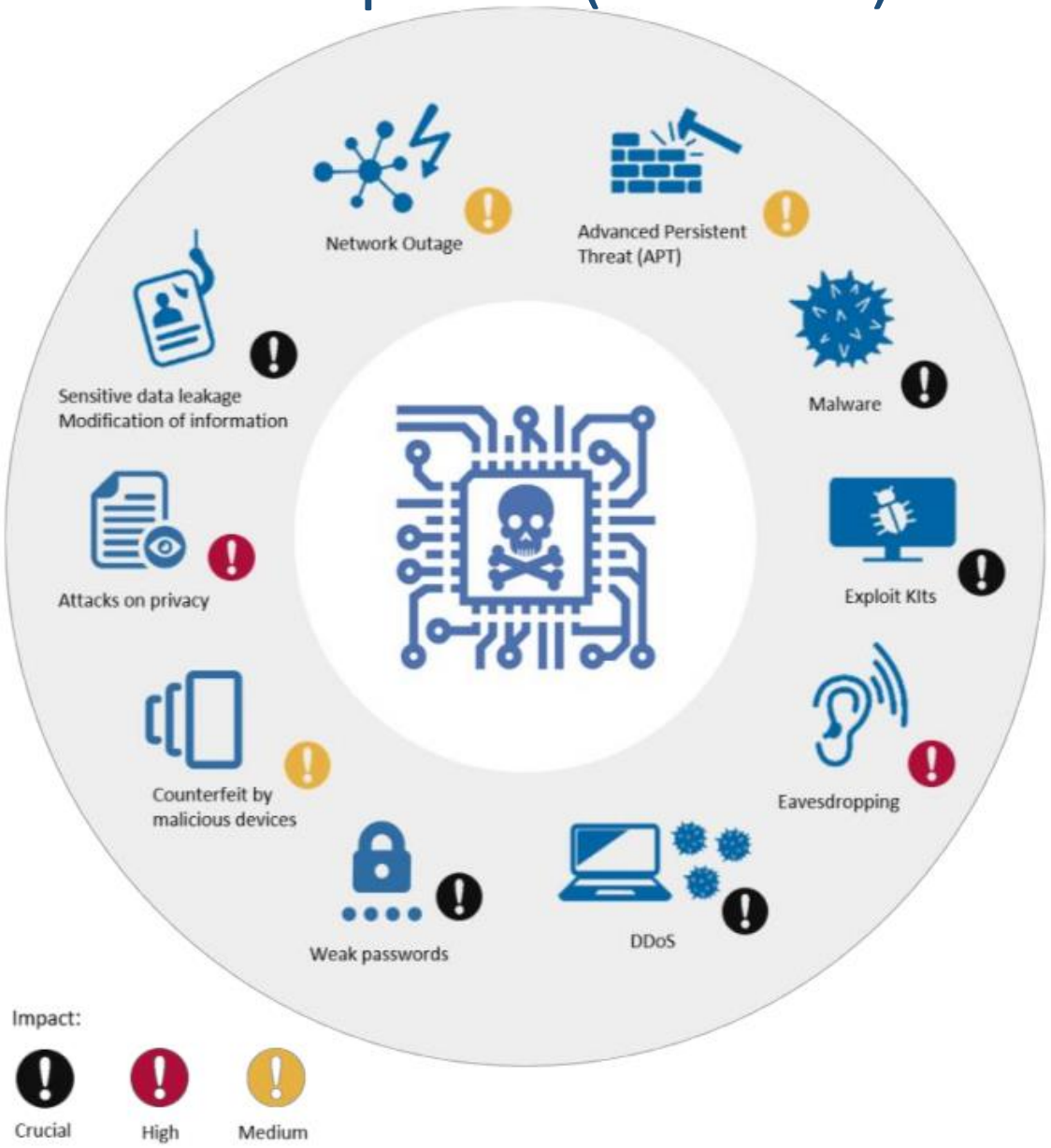
ENISA IoT threat taxonomy (Nov. 2017)

Consistent with the threat taxonomy released in Jan 2016, but easier to navigate and IoT focused

ENISA IoT threat taxonomy (Nov. 2017)



IoT threats impact (ENISA)



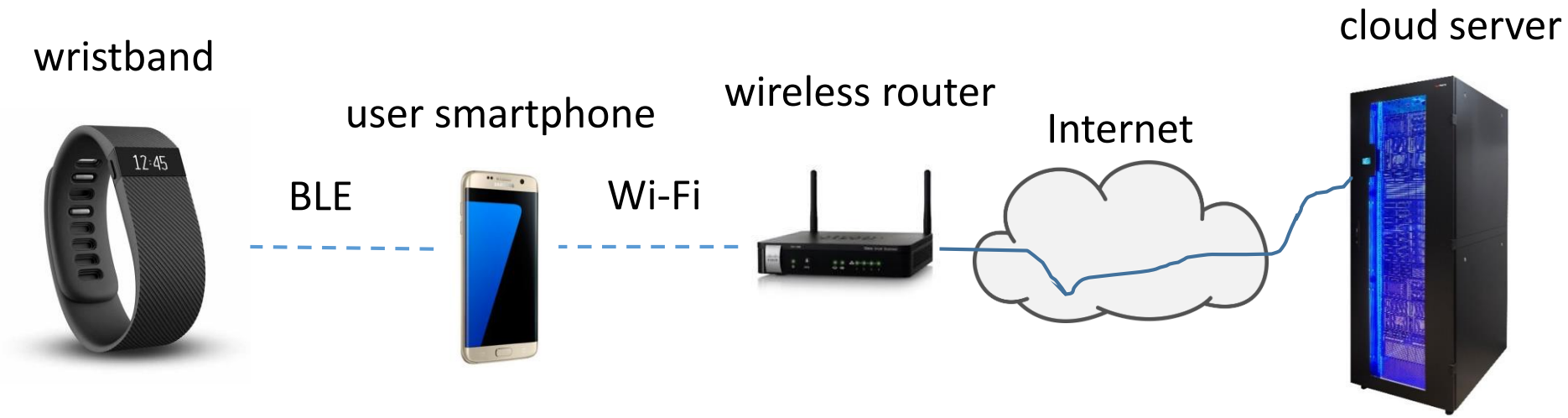
OWASP IoT Project

- OWASP (Open Web Application Security Project) – non-for profit initiative, focused on improving software security
- IoT project “designed to help manufacturers, developers, and consumers better understand the security issues associated with the Internet of Things, and to enable users in any context to make better security decisions when building, deploying, or assessing IoT technologies.”

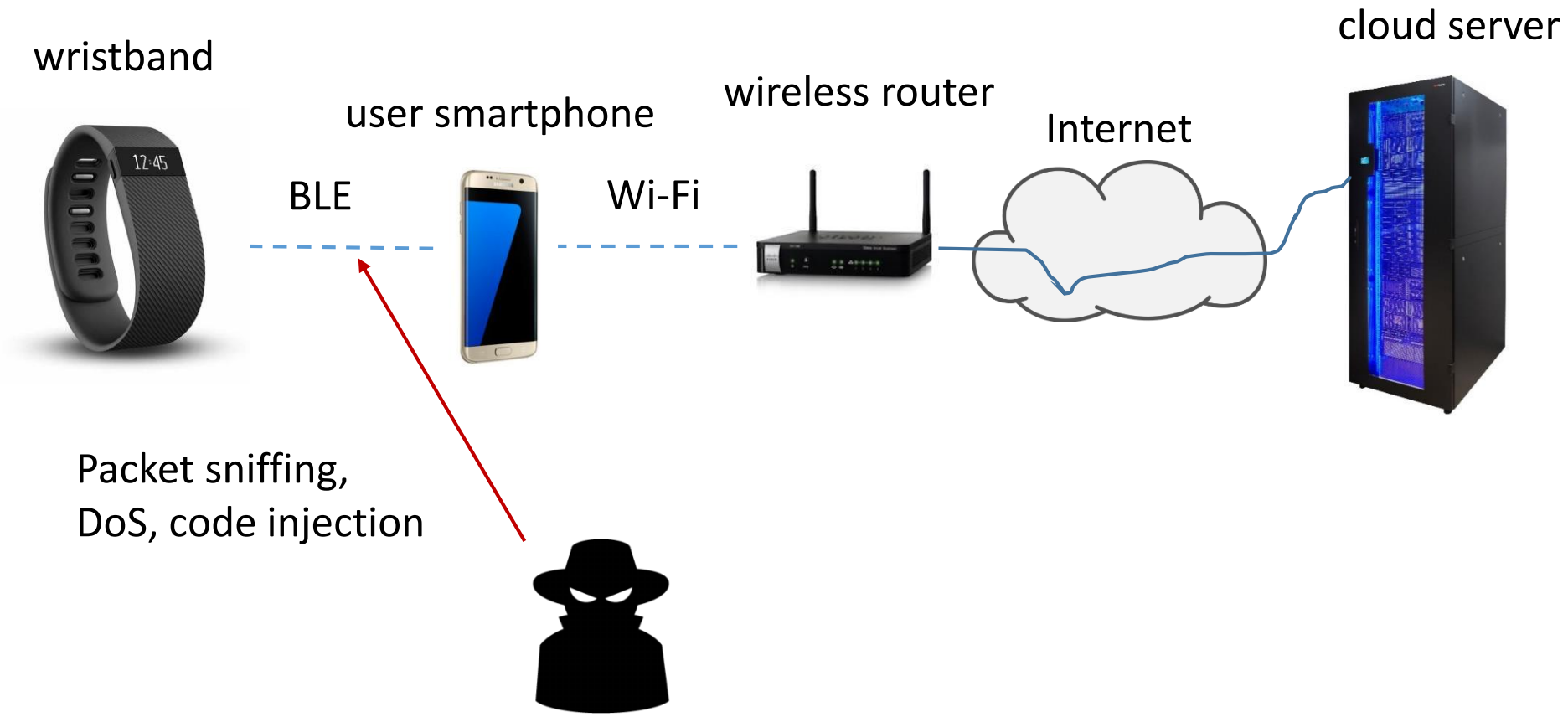
OWASP IoT Attack Surface

- Ecosystem (general)
- Device memory (sensitive data)
- Device physical interfaces (privileges, debug ports, FW extraction, etc.)
- Device web/mobile interface/app (credential management)
- Device firmware (data exposure, vulnerable services)
- Device network services (injection, DoS, FW OTA block, replay attacks, buffer overflows, etc.)
- Admin interface (credentials mgmt., 2FA)
- Local data storage
- Cloud interface
- Privacy
- Several others

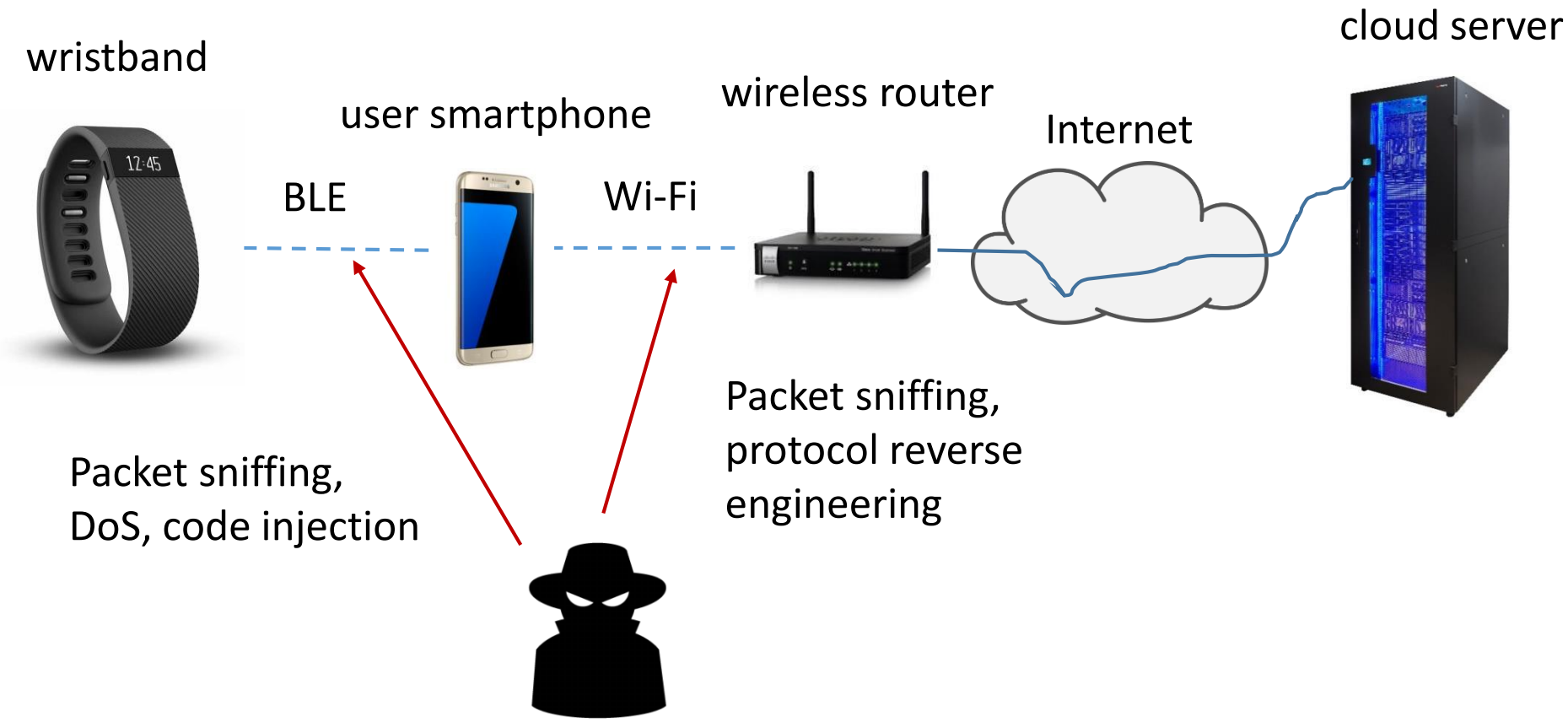
Example – fitness tracking systems



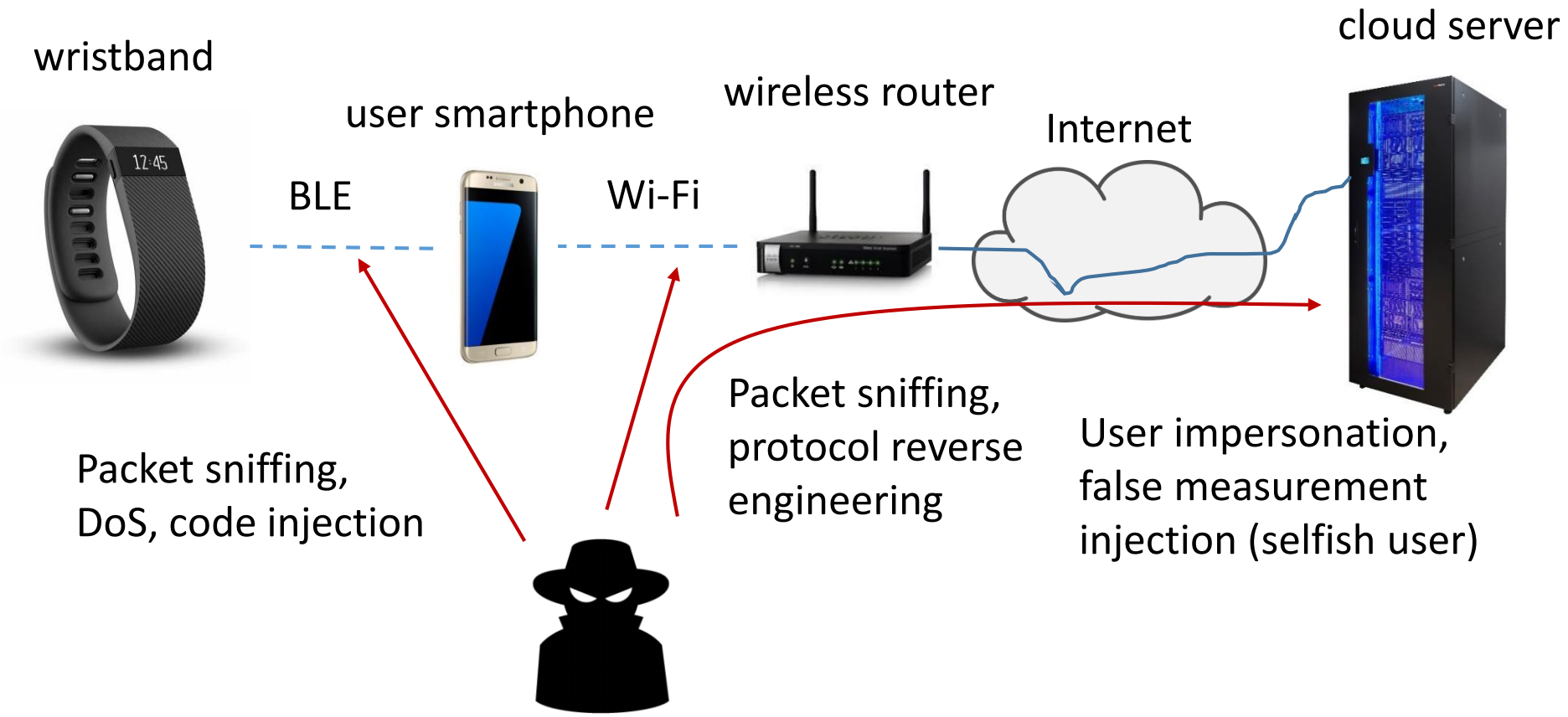
Example – fitness tracking systems



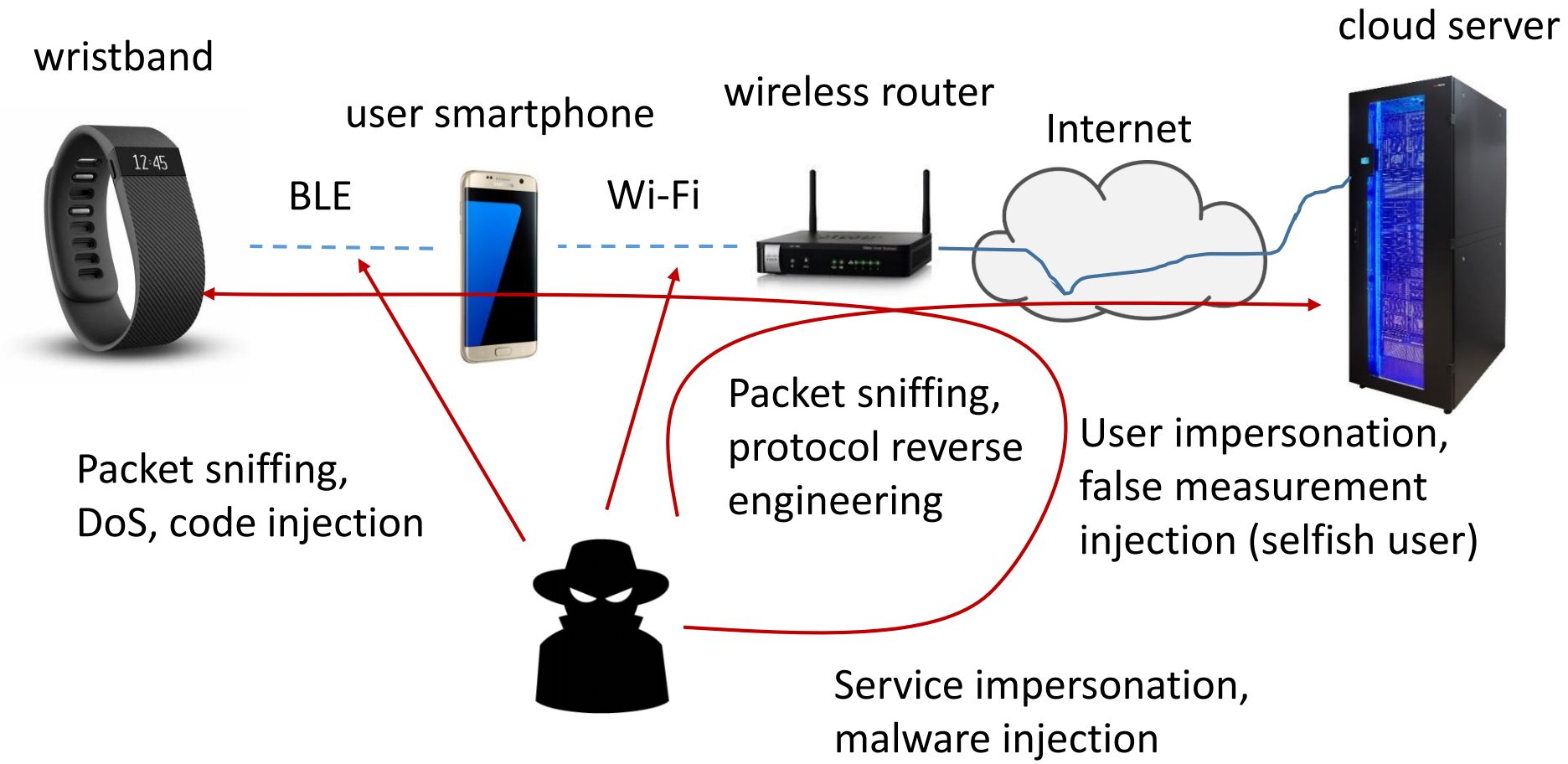
Example – fitness tracking systems



Example – fitness tracking systems



Example – fitness tracking systems



How to mitigate such a multitude of risks?

Efforts needed across the entire end-to-end system

- Hardware (device cannot be compromised through direct physical access)
- Code (firmware/software trustworthy)
- Data not sent in plaintext (encryption)
- Service semantics (communication robust to DoS, replay attacks/account hijacking, etc.)

Hardware security

How to ensure that keys stored on a device cannot be extracted easily?

How to ensure the boot process remains untampered even when the application might get compromised?

Note: might not be possible to make device completely safe against attacks, yet damage can be limited through several approaches.

One-Time-Programmable (OTP) memory

- Permanently programmed memory cells with strong security (state-of-the-art today)
- Access tightly controlled, difficult to reverse engineer
- Programmed during IC or subsequent system manufacturing.
- With some, possible to destroy stored keys in response to tamper attempts.
- Secure key injection typically used to deploy randomly generated keys.
- Security may be maximized if the IC generates its own keys using hardware designed into the device.

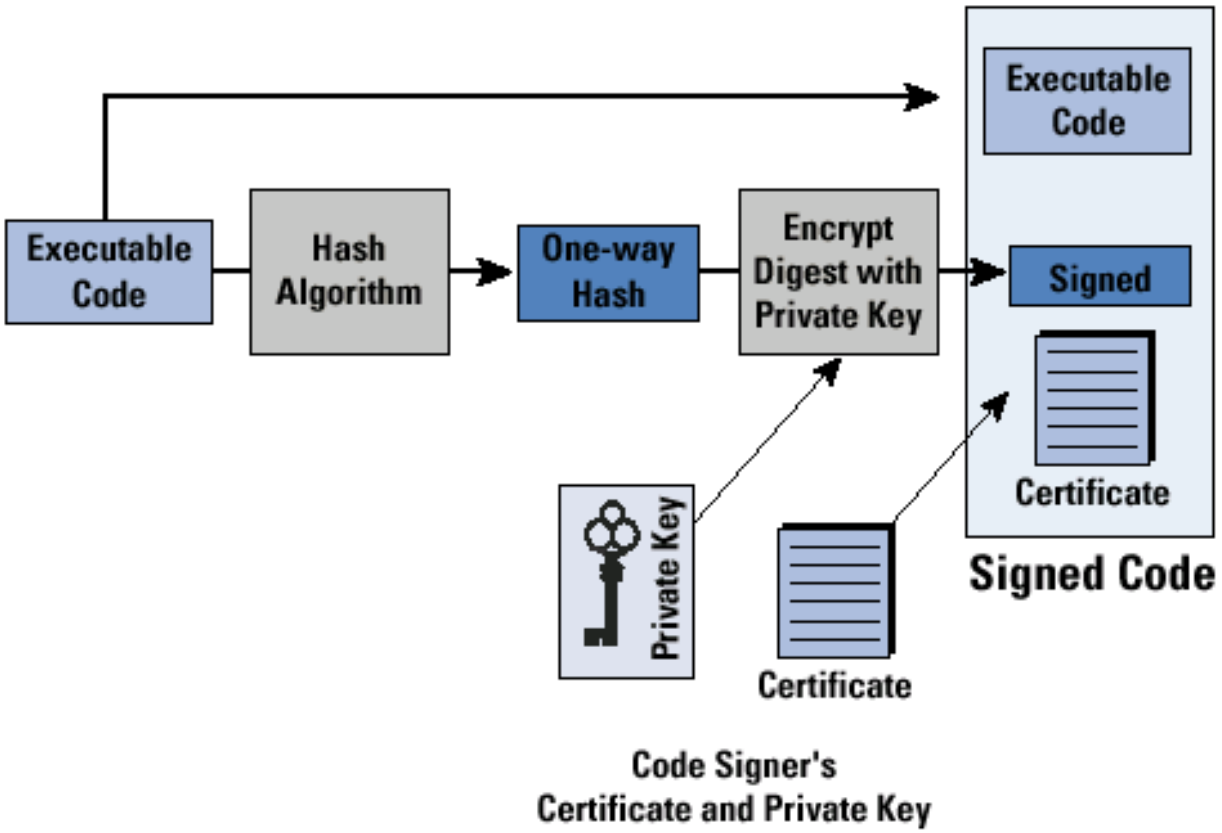
Electrically Erasable Programmable Read Only (EEPROM) memory

- Used as separate key/bootloader memories.
- Used in multichip modules, drawing cryptographic boundary around the subsystems.
- Key injection used to provision secret keys.
- Possible to bar EEPROM writing instructions from app code.
- Can disable EEPROM writing completely via configuration fuses - even firmware updates will not be allowed

Code signing

- Devices should be able to verify whether the code pushed by a developer can be trusted (think about firmware updates)
- Typically a cryptographic hash produced to confirm the authenticity and integrity of the software
- Key idea: map data of any size to a fixed string, using a one-way function (i.e. difficult to invert).
- Public-key-infrastructure (PKI) used to provide assurance about who signed the code. Signature embeds a certificate issued by a certificate authority (CA).

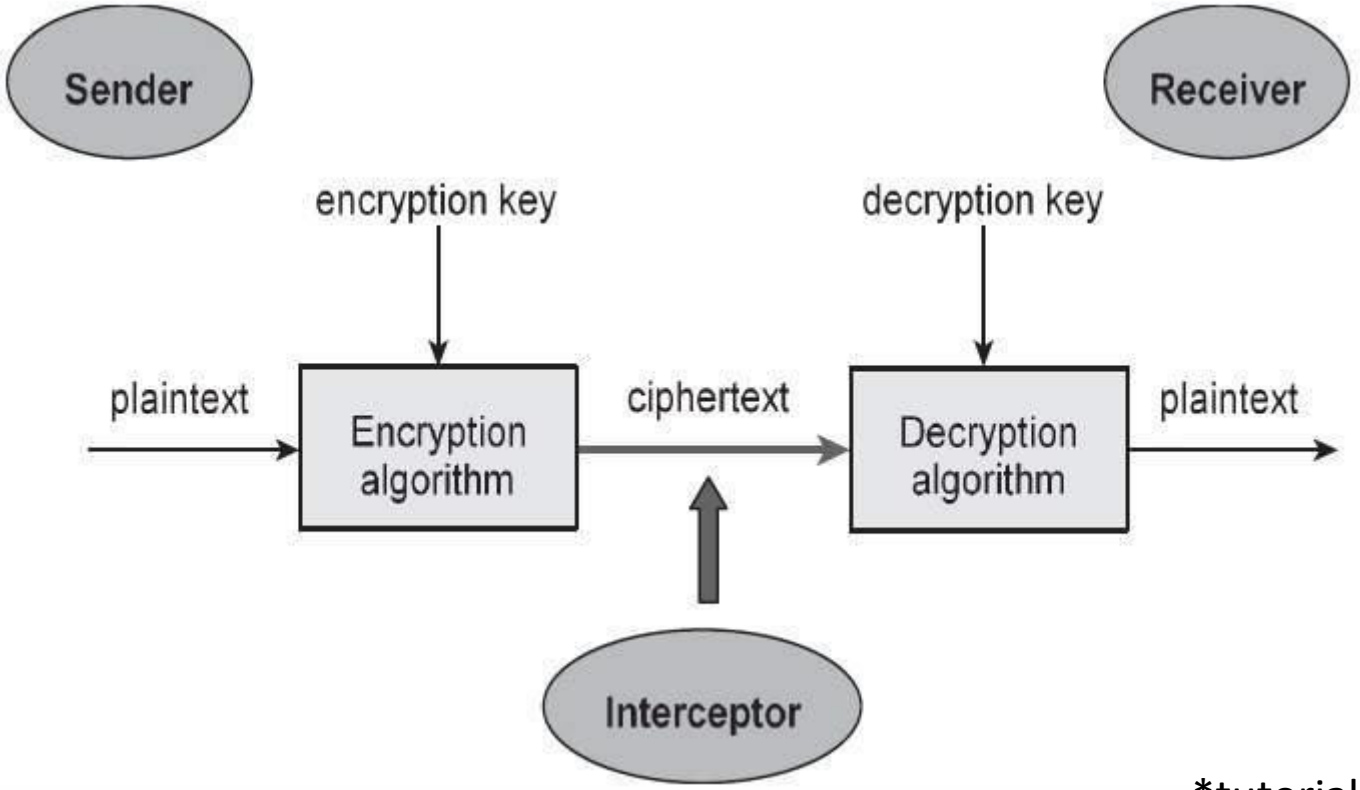
Code signing



*Cisco

Network security - encryption

Simple encryption model – transforming plaintext into cyphertext using some key; attacker intercepts messages, tries to figure out key



*tutorialspoint.com

Encryption principles - secrecy

- The secrecy is within the key - its length is a major design issue (e.g. locks: 2 digits = 100 combinations)
- The longer the key, the higher the work factor!
- Work factor for breaking the key by exhaustive search is exponential in key length.

Encryption principles - redundancy

- Encrypted message must contain some information that is not needed to understand the original message
- Even if the key is long, an attacker may generate fake messages when the ciphertext is short by using random numbers
- Adding redundancy decreases the likelihood of generating a valid message,
- But at the same time makes it easier to break original messages overheard

Encryption principles - freshness

- Some method is needed to avoid replay attacks (i.e. allow to verify that the message was recently sent)
- One possible solution is to include a timestamp in every message that is valid only for a limited time *T* (e.g. 10 seconds)
- Receiver keeps a message for *T* seconds, compares new messages to previous ones, and filters out duplicates.
- Messages older than *T* seconds will be discarded as considered too old.