WEBINAR

Implementing the 5 Pillars of IT Security for MQTT

HIVEMQ

WELCOME

Magi Erber

Product Manager @HiveMQ

@ErberMagi

linkedin.com/in/margaretha-erber/

Georg Held

Software Developer @HiveMQ

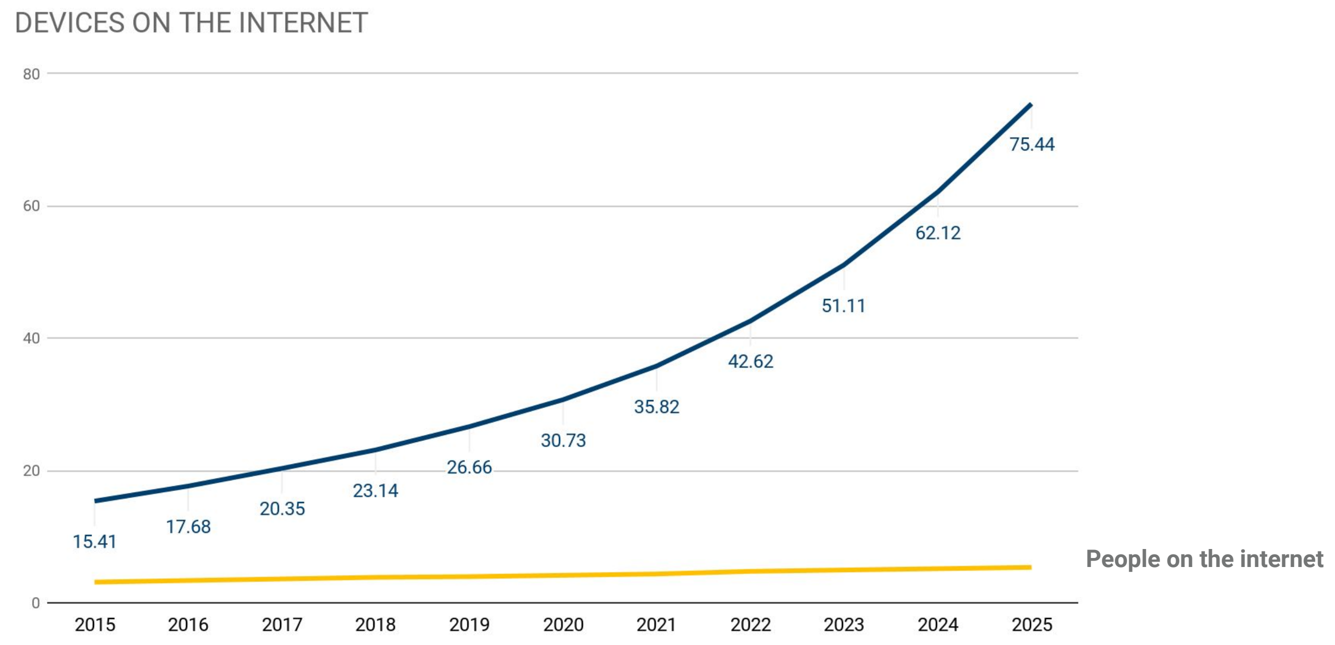
github.com/sauroter/

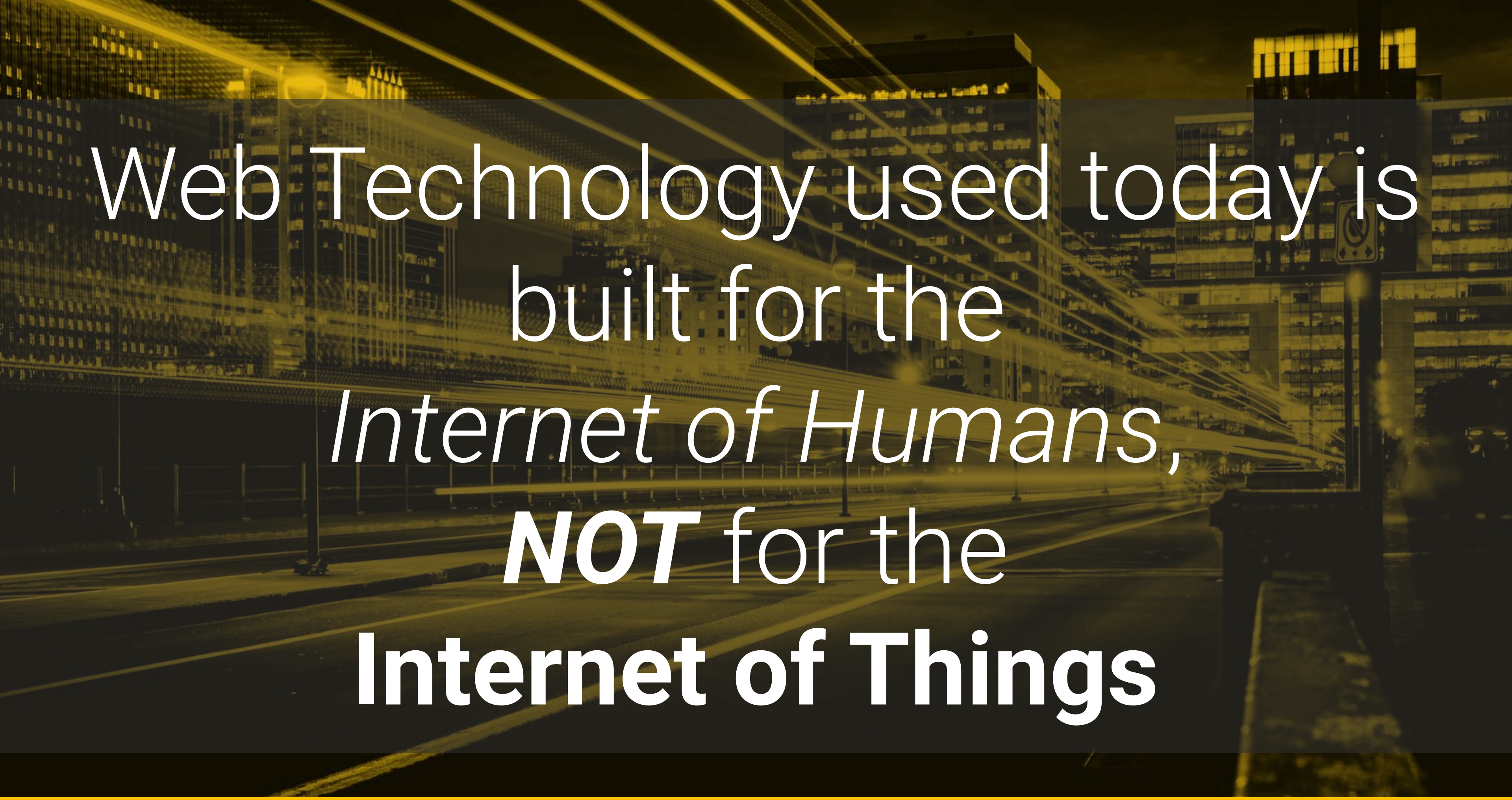
linkedin.com/in/sauroter/

HIVEMQ

As we speak millions of things are newly connected to the internet

The Internet of Things is HUGE





Web Technology used today is built for the *Internet of Humans*, ***NOT*** for the **Internet of Things**

HIVEMQ

Technical IoT Challenges

- **Scalability:** Massive scalability required for millions of devices
- **Instant data delivery:** critical systems need reliable and instant data transfer
- **Unreliable networks:** excellent customer experience for IoT apps and devices

We need **open standards** designed for the **Internet of Things**

MQTT

What is MQTT?

- (I)IoT Messaging Protocol
- Created for extreme scale and instant data exchange
- Publish/Subscribe based architecture
- Easy on the device side, pushes all implementation complexity to the server
- Built for machines and constrained devices (binary, data agnostic)
- Designed for reliable communication over unreliable channels

MQTT Use Cases

Connected Car

IIoT / Industry 4.0

Logistics

Telecommunication

IoT Messaging Middleware

MQTT Use Cases

Push Communication

Reliable Communication over unreliable networks

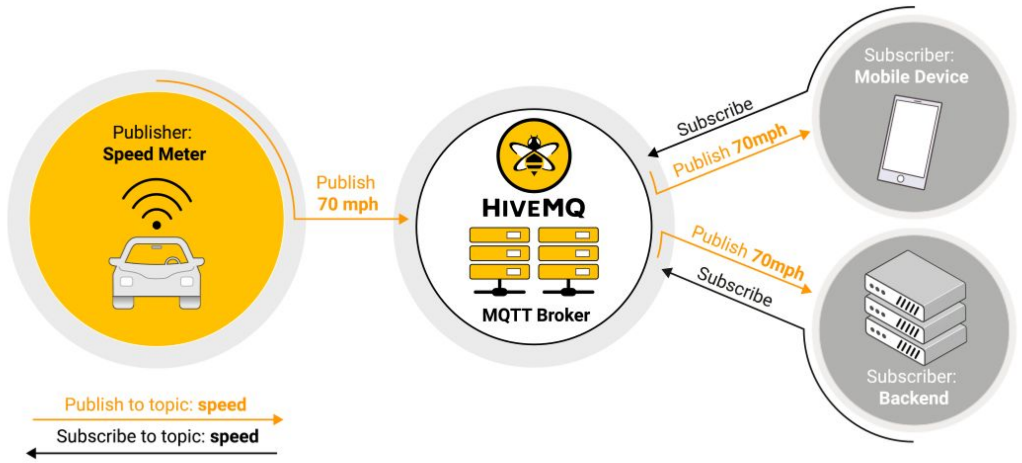
Constrained Devices

Low Bandwidth and High Latency

Industrial Message Bus

MQTT

- Lightweight protocol on top of TCP/IP
- Publish / Subscribe pattern using topics
- De-coupling of sender and receiver



Security Challenges for IoT Use Cases

Challenge 1 - Sensible Data

- Connecting things exposes sensible data to the internet

$\longrightarrow$

- Data breach would damage the reputation of your business

  Bad press can ruin your business (unit)

Challenge 2 - Control over IoT Devices

- **Compromised devices for daily usage can be extremely dangerous**

  → Attacker could get control over device

- **Compromised devices can open access to company infrastructure**

  → Point of entry for fraudsters to steal corporate secrets

Challenge 3 - Legal or Corporate Regulations

- **Legal regulations for data privacy and safety**

  → Software must be compliant to a bunch of legal regulations like GDPR, CCPA, ...

- **Corporate compliance policies**

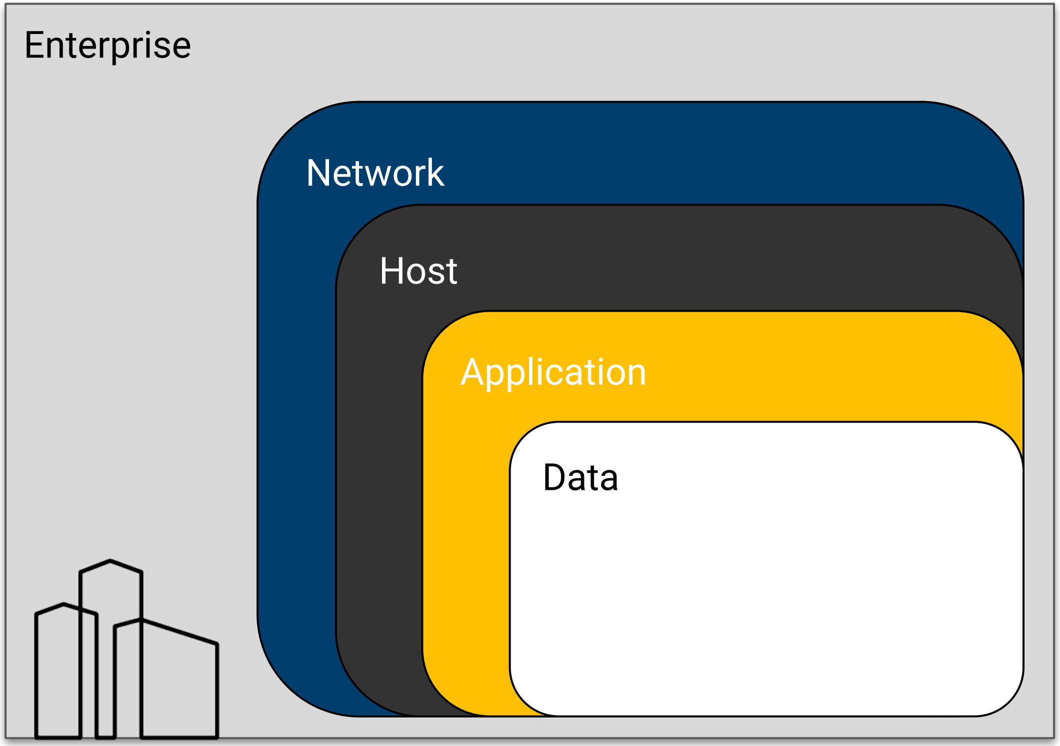
  → In addition software must be compliant to corporate specific regulations

Challenge 4 - Customer Experience

- **Customer experience for IoT apps and devices must be excellent even when security is in place**
  - → Combination of high grade security and excellent user experience
- **Devices and apps must be easy to program and maintain, complexity should be at the broker not on the device**
  - → Broker is easier to update than physical devices

Security is a key concern for any application

Multiple Security Layers



5 pillars of IT Security

5 Pillars of IT Security

Confidentiality

Information is only available to authorized parities (and to no one else)

Integrity

Information can only be put into the system or changed by authorized parties (and no one else)

Availability

The system can be accessed by authorized parties at any time

Authenticity

Information is associated with a source

Non-Repudiation

Actions are associated with a source after the fact

Confidentiality

Confidentiality

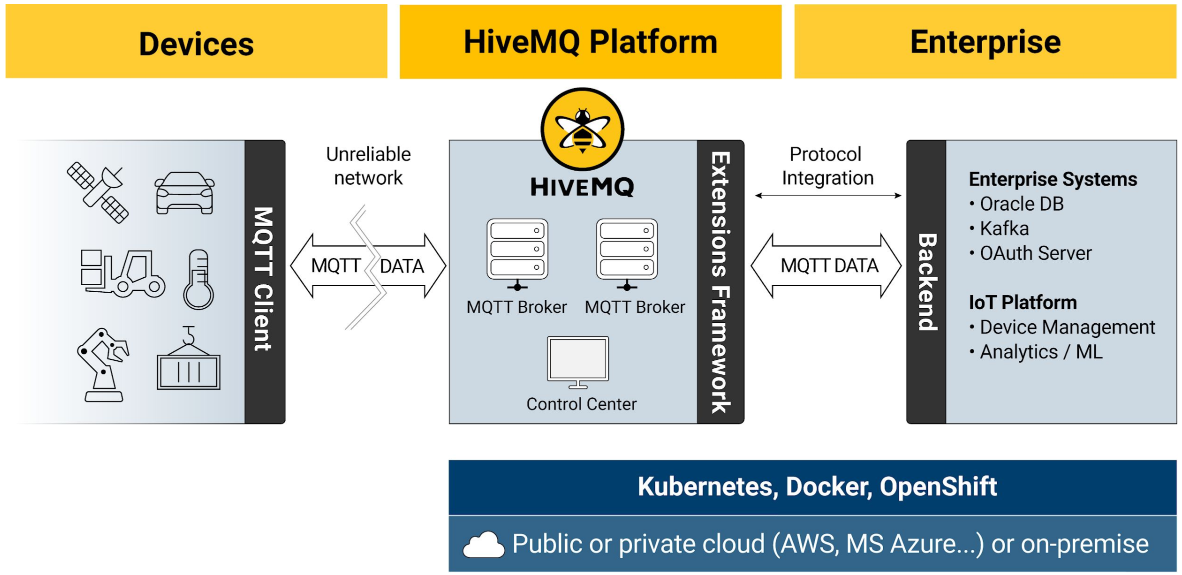
Information is only **available to authorized parties** (and to no one else)

- This is the thing we most often care about.
- We don't want competitors to get our data.
- We don't want to be liable for data loss (GDPR).
- We want to validate the trust placed in us by our customers.

Confidentiality

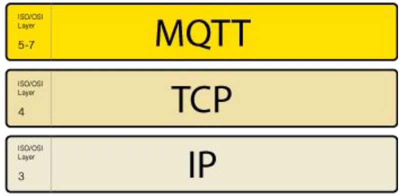
- Trusted environments
- Transport encryption
- Mutual Authentication
- Subscriber Authorization

Trusted Environments



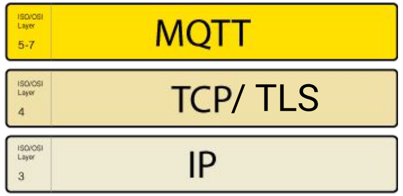
- Run broker and devices in **trusted environments**
- Broker often works as **interface** between public untrusted environment and secure backend
- If devices are not under your control, **don't trust them unconditionally**

Transport Encryption



- MQTT is based on **TCP / IP Stack**
- Listens on **Port 1883** for communication

- TCP connection can be **secured by TLS**
- Listens on **Port 8883** for secure communication



TLS - Transport Layer Security

Cryptographic protocol

Secure communication and authenticated channel between server and clients

TLS Handshake initiates TLS session

TLS secured communication can't be eavesdropped by anyone

Prevents Man-in-the-Middle attacks

Provision and Revocation of Certificates

Provision of certificates

- Needs a planned **provisioning and certificate lifecycle process**
- Deployed **PKI (public key infrastructure)**

Revocation of certificates

- Needed as soon as certificates **can't be trusted anymore**
- **Certificate Revocation List** (CRL)
- OCSP for **online certificate validation**
- Use a management system like Hashicorp Vault

Transport Encryption - Best Practices

- Use transport encryption (TLS)
- Use certificates from trusted CAs
- Use highest TLS version and secure cipher suites

Cipher Suites

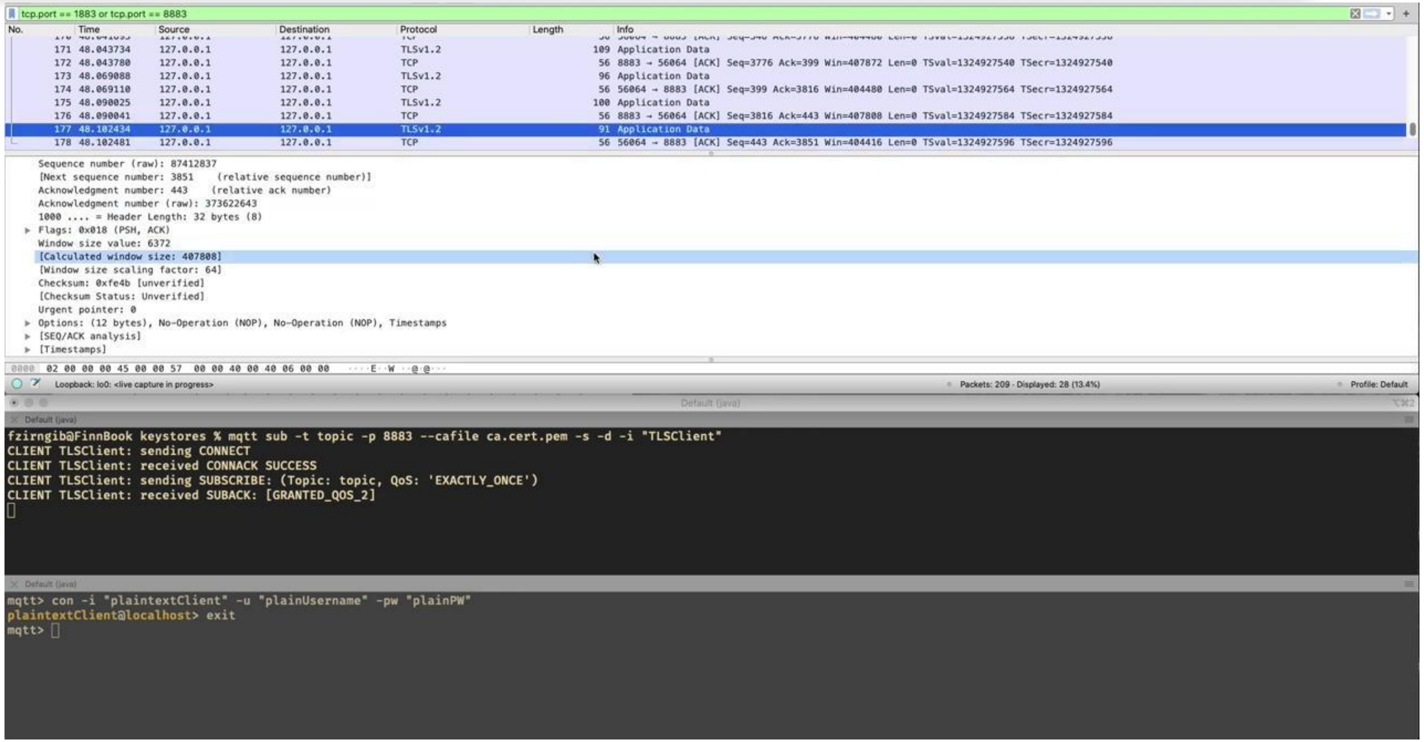
`TLS_AES_256_GCM_SHA384`

- High bit length (big numbers)
- Proven Algorithms (AES)
- Safe Modes (GCM)

For TLS <= 1.2

- Ephemeral Diffie Hellman key exchange (e.g. `TLS_ECDHE_…`)

Transport Encryption - Example



tcp.port == 1883 or tcp.port == 8883

```
No.    Time        Source       Destination   Protocol   Length  Info
  171 48.043734   127.0.0.1    127.0.0.1     TLSv1.2      109 Application Data
  172 48.043780   127.0.0.1    127.0.0.1     TCP           56 8883 → 56064 [ACK] Seq=3776 Ack=399 Win=407872 Len=0 TSval=1324927540 TSecr=1324927540
  173 48.069088   127.0.0.1    127.0.0.1     TLSv1.2       96 Application Data
  174 48.069110   127.0.0.1    127.0.0.1     TCP           56 56064 → 8883 [ACK] Seq=399 Ack=3816 Win=404480 Len=0 TSval=1324927564 TSecr=1324927564
  175 48.090025   127.0.0.1    127.0.0.1     TLSv1.2      100 Application Data
  176 48.090041   127.0.0.1    127.0.0.1     TCP           56 8883 → 56064 [ACK] Seq=3816 Ack=443 Win=407808 Len=0 TSval=1324927584 TSecr=1324927584
  177 48.102434   127.0.0.1    127.0.0.1     TLSv1.2       91 Application Data
  178 48.102481   127.0.0.1    127.0.0.1     TCP           56 56064 → 8883 [ACK] Seq=443 Ack=3851 Win=404416 Len=0 TSval=1324927596 TSecr=1324927596
```

```
    Sequence number (raw): 87412837
    [Next sequence number: 3851    (relative sequence number)]
    Acknowledgment number: 443    (relative ack number)
    Acknowledgment number (raw): 373622643
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 6372
    [Calculated window size: 407808]
    [Window size scaling factor: 64]
    Checksum: 0xfe4b [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]

0000  02 00 00 00 45 00 00 57  00 00 40 00 40 06 00 00   ····E··W ··@·@···
```

Loopback: lo0: <live capture in progress>   Packets: 209 · Displayed: 28 (13.4%)   Profile: Default

Default (java)

```
fzirngib@FinnBook keystores % mqtt sub -t topic -p 8883 --cafile ca.cert.pem -s -d -i "TLSClient"
CLIENT TLSClient: sending CONNECT
CLIENT TLSClient: received CONNACK SUCCESS
CLIENT TLSClient: sending SUBSCRIBE: (Topic: topic, QoS: 'EXACTLY_ONCE')
CLIENT TLSClient: received SUBACK: [GRANTED_QOS_2]
```

Default (java)

```
mqtt> con -i "plaintextClient" -u "plainUsername" -pw "plainPW"
plaintextClient@localhost> exit
mqtt>
```

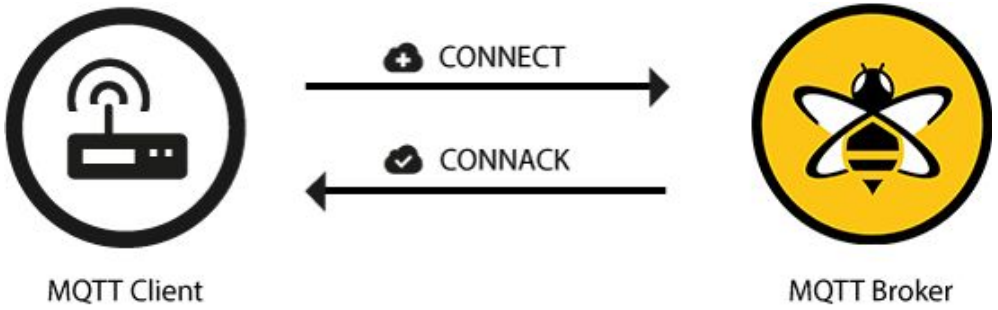
Payload Encryption

On very constrained devices transport encryption could be not possible!

- Use payload encryption instead
- At least hash or encrypt password of connecting client
- Every clients needs to have key & secret
- *BUT!: It leaks metadata*

Mutual Authentication

Authentication verifies whether a person, device or application is who they say they are



Authentication can take place on the **Transport Layer** and on the **Application Layer**

Mutual Authentication using Certificates

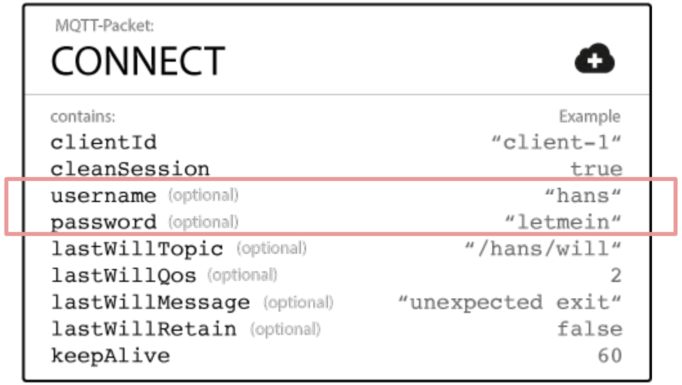
Transport Layer Authentication

- **Mutual Authentication** of broker and client using the presented **certificate at TLS** handshake
- Authentication takes place **before a secure communication channel is established**

Application Layer Authentication / Authorization

- Client is granted **permissions based on certificate information**

Client Authentication (Identity and Access Management Systems)


MQTT-Packet:
CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

Caution:
Not all brokers support a pluggable authentication and authorization system!

- Different **external systems** can be used to authenticate clients at a broker
- Client provides **authentication data in the CONNECT packet**
- Broker **looks up the authentication data** in the connected external systems
- External authentication systems:
  - LDAP
  - OAuth2.0
  - Databases
  - ACL
  - ...

Mutual Authentication - Example

Subscriber Authorization

Authorization provides access rights to a resource

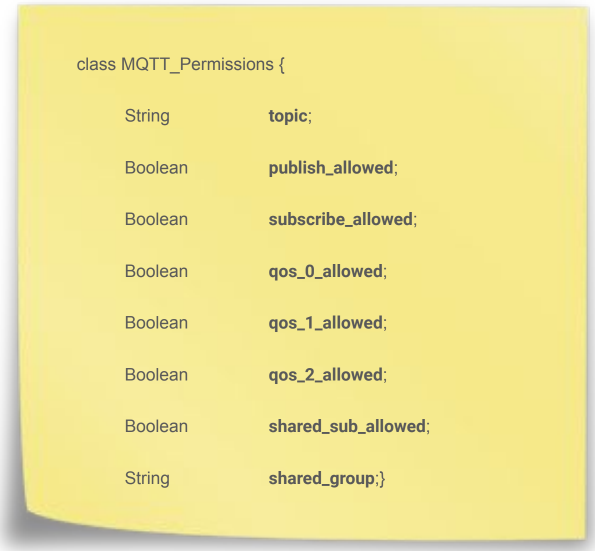
- Without authorization **every client is allowed to subscribe to all topics**
- Clients should **only get the data they are allowed to get**
- **Permission structure must match topic structure**
- Tip: Use **client identifiers in topics** where possible
- *Be careful with wildcard subscriptions!*

HIVEMQ

Subscriber Authorization

Permission includes:

- Allowed **Topic** (exact topic or topic filter including wildcards)
- Allowed **Operation** (Subscribe, Publish, both)
- Allowed **QoS** (0, 1, 2, 0-1, 1-2, all)
- Allowed **specific operations** (retained messages, shared subscriptions)



```
class MQTT_Permissions {
        String          topic;
        Boolean         publish_allowed;
        Boolean         subscribe_allowed;
        Boolean         qos_0_allowed;
        Boolean         qos_1_allowed;
        Boolean         qos_2_allowed;
        Boolean         shared_sub_allowed;
        String          shared_group;}
```

Integrity

Integrity

Information can only be **put into the system or changed by authorized parties** (and no one else)

- Devices make decisions based on data.
- Customers expect correct data.
- Deployment teams take actions based on data.

Integrity

- Transport Encryption
- Mutual Authentication
- Publisher Authorization
- Broker / Device access security

Publisher Authorization

🚫 Blocklist

- Everything that is not explicitly denied, is allowed
- Offers protection as long as all restrictions are known for all clients
- Threats are only stopped after list has been updated

🔅 Whitelist

- Everything that is not explicitly allowed, is denied
- High maintenance effort
- Access is only granted after list has been updated

Authorization of Publishers - Example

Access Security (Operating System)

- Keep libraries and software updated
- Every connection should pass at least one Firewall
- Disallow root access and use SSH keys for SSH
- Use file system permissions (`chmod -R X00 ...`)
- Install a intrusion detection and prevention system
- Use SELinux

Access Security (Tamper proofing devices)

- Remove JTAG access
- Use signed firmware
- Secure access to X.509 Certificates (read only)
- Endpoint Security
- …

Availability

Availability

The system can be **accessed by authorized parties at any time**

- Excursion: A cautious tale of the most secure software system.
- Only a used system generates any value
- Customers pay for the availability of a system

Availability

- MQTT software selection
- Incident management
- Support systems

Criteria for selecting the right MQTT Broker

- Performant, scalable and high available broker
- Track Record and Reputation of the broker vendor
- Longevity: Long Term Support for broker software
- Compliance to the entire MQTT specification
- Monitoring of broker and tracing of devices
- Pluggable authentication & authorization system
- (D)DoS Detection
- Overload Protection
- Support of TLS
- Professional support

HIVEMQ ENTERPRISE

mosquitto

AWS IoT

RabbitMQ

Overload Protection - Example

Criteria for selecting the right MQTT Client

- Efficient and reliable implementation of the whole standard
- Supports all MQTT security features
- Longevity: Long Term Support for client software

Criteria for selecting the right Integration Systems

- Highly proven and production grade third party systems
- Track Record and Reputation of the vendor
- Longevity: Long Term Support for software

Incident Management

Clear responsibilities for incident management inside your organisation

Defined processes with a multi tiered escalation

Direct access to subject matter experts via vendor support contracts.

Regular trainings and rehearsals

High skill level of responsible team

Regular Backups of the system

Support Systems

Load Balancers

Scalable Infrastructure

Automated monitoring and flexible logging handling

Alerting Systems

IDS / IPS System with the right data source (Access Log, ip based)

Other Pillars

Authenticity

Information is associated with a source

-> Payload Signing

Non-Repudiation

Actions is associated with a source after the fact

-> Audit Log

Auditability

The system generates and stores a paper trail

-> Access Log

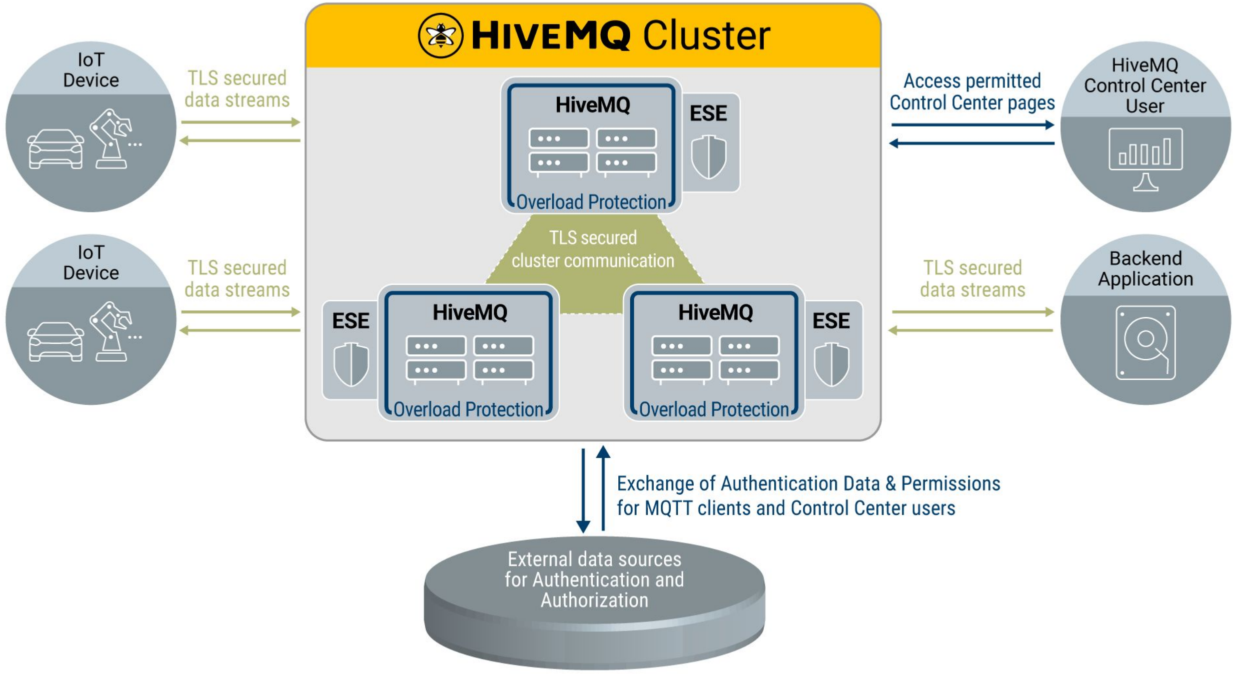
Privacy

Personal data is not leaked

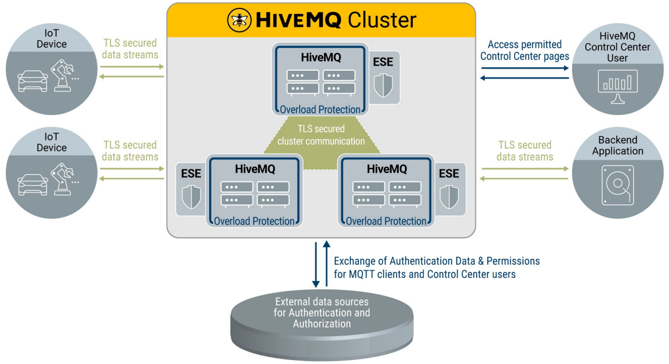
-> Must be implemented in the infrastructure

Security is a **key concern** for any IoT application. HiveMQ implements the security features required for **safe and secure enterprise IT and OT deployments.**

HiveMQ Security Architecture


HIVEMQ Cluster
IoT Device
TLS secured data streams
IoT Device
TLS secured data streams
HiveMQ
ESE
Overload Protection
TLS secured cluster communication
ESE
HiveMQ
Overload Protection
HiveMQ
ESE
Overload Protection
Access permitted Control Center pages
HiveMQ Control Center User
TLS secured data streams
Backend Application
Exchange of Authentication Data & Permissions for MQTT clients and Control Center users
External data sources for Authentication and Authorization

HiveMQ Security Architecture


HIVEMQ Cluster
IoT Device
TLS secured data streams
IoT Device
TLS secured data streams
HiveMQ
ESE
Overload Protection
TLS secured cluster communication
ESE
HiveMQ
Overload Protection
HiveMQ
ESE
Overload Protection
Access permitted Control Center pages
HiveMQ Control Center User
TLS secured data streams
Backend Application
Exchange of Authentication Data & Permissions for MQTT clients and Control Center users
External data sources for Authentication and Authorization

- Pluggable Authentication and Authorization System
- Prebuilt Security Extension
- TLS secured communication
- Overload Protection and (D)DOS detection
- Fine grained permission system for MQTT clients and HiveMQ Control Center users
- Chaining of auth mechanisms
- Default Deny-All behaviour
- Integrated monitoring system and over 800 metrics
- 24/7 professional support

HIVEMQ
Enterprise Security Extension

HiveMQ Enterprise Security Extension

HIVEMQ
Enterprise Security Extension

- Central management for **IoT device and HiveMQ Control Center authentication and authorization**
- Flexible and easy **integration with multiple external authentication systems and data sources** (e.g. databases, LDAP, OAuth 2.0)
- High **Scalability and reliability**
- Default **Whitelisting Concept**
- **Access log** (rolling on daily basis)
- Provides maximum flexibility in defining authorization rules

ANY QUESTIONS?

Reach out to community.hivemq.com

THANK YOU

Stay updated on upcoming webinars

Subscribe to our Newsletter!

HIVEMQ