Threat models and moving target defense for the CoAP messaging protocol.

Carolyn Talcott

LAP 24

The Problem

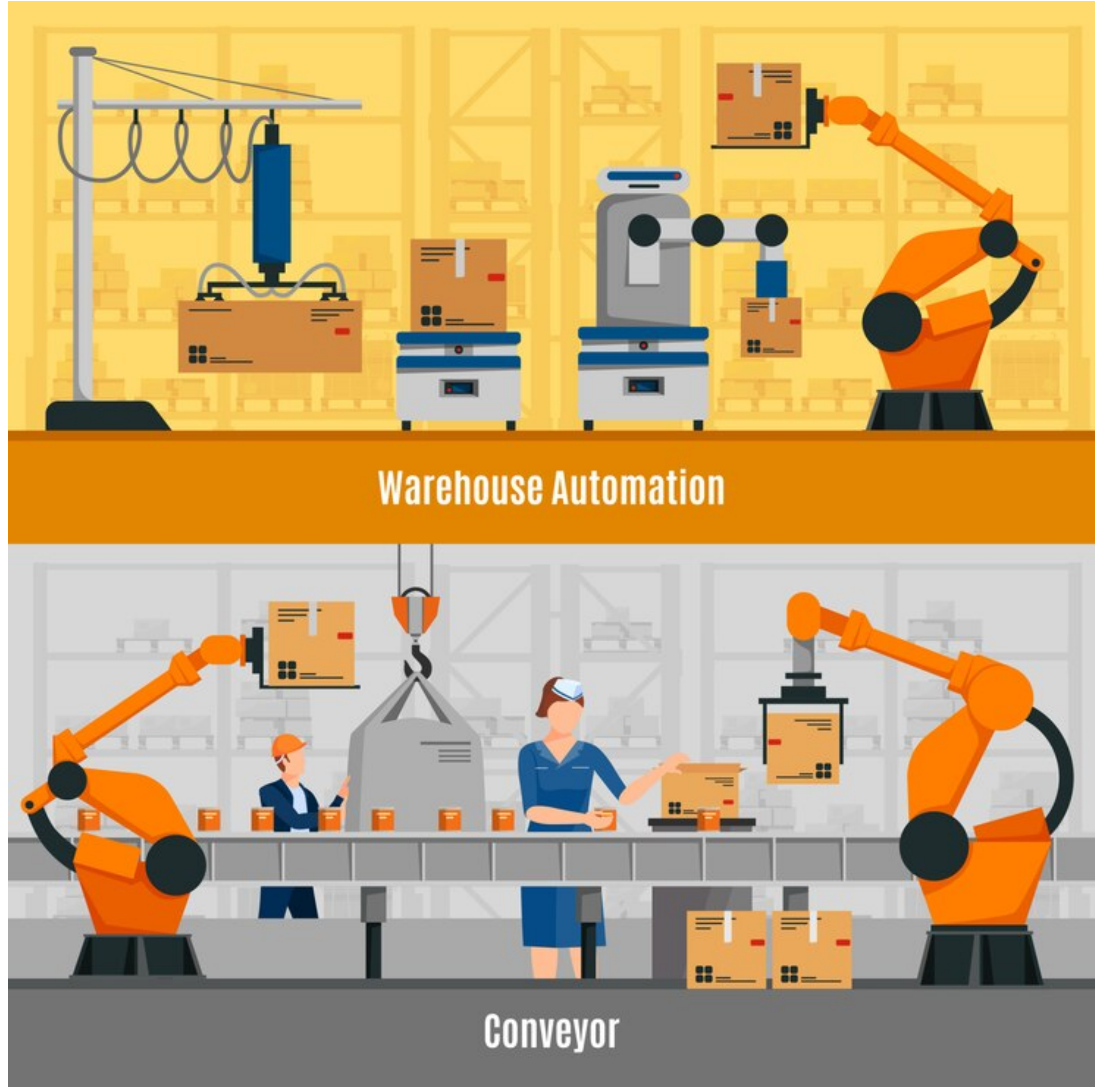
- Networked applications based on Internet of Things (IoT) provide many services
  - some for convenience or entertainment
  - some safety critical
    - smart buildings
    - infrastructure (electrical grid, movable bridges)
    - manufacturing and process control,
    - medical devices, ...
- Applications run on resource limited devices and communicate over unreliable, bandwidth limited networks.
- Many IoT devices are mass produced, enhancing vulnerabilities.

IoT Communication

- IoT Applictions use light-weight messaging protocols such as Zigbee (Wirelss Mesh), MQTT (PubSub), CoAP (HTTP).
- Communcations often concern sensor readings or actuator commands
  - may be order and/or time sensitive
  - may affect the real world
    - door lock/unlock
    - heat/water/light control
    - control of chemical, manufacturing process
- Attacks may have physical consequences

Reasoning about IoT Communication

- Reasoning about IoT messaging protocols may involve physical properties and can be application specific, different than traditional security protocols
  - no fixed set of messages to consider
  - no standard set of properties to check
  - some messages are more vulnerable than others



Dialects: a Moving Target Defense

- A dialect is a wrapper that uses lightweight transformations to obfuscate communications.
- Moving target: transformation parameters change frequently and unpredictably.
- A dialect should foil attacker attempts to unovfuscate or spoof ensuring messages are
  - only processable by the intended target
  - only sent by the claimed source
- Formally modeled as a transformation on the theory specifying the underlying protocol

Plan

- Overview of the CoAP messaging protocol and specification in Maude
- Attack Models
- A CoAP Dialect
  - Definition
  - Properties
  - Attack Analysis
- Application layer
  - Symbolic attack search
  - Moving bridge case study

Constrained Application Protocol (CoAP)

- Constrained Application Protocol (CoAP) is a protocol designed for constrained networks and devices, defined in RFC 7252.
- CoAP uses an HTTP like request/response interaction model
    - client sends requests to access and modify some server resource
    - server processes requests and sends response
- A adevice may play both client and server roles
- CoAP Runs on an unreliable underlying network (UDP) Uses CON vs NON mode message types to provide some reliability control

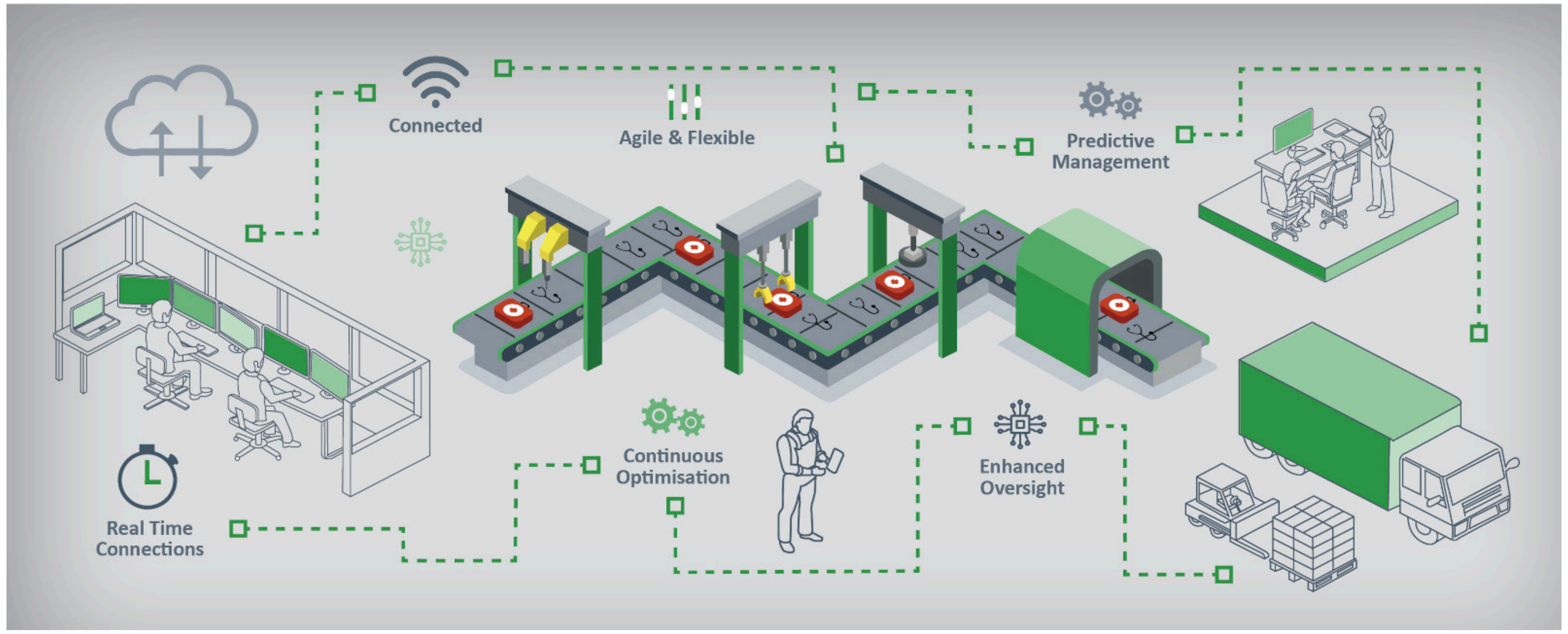
```
client    server(door)
  |            |  lock
  | - PUTDL-> |
  | <-2.04-   | unlock
  |            |
```

```
client    server(door)
  }            |  lock
  | - PUTDL-@ |  message dropped
  |   ...      |  time passes
  | - PUTDL-> |  retry
  | <-2.04-   | unlock
  |            |
```

CoAP specification as a rewrite theory - briefly

- Messages: m(tgt,src,content)
  - content: request or response
- Execution State: a set of endpoints and a network holding delayed messages, msg @ d, in transit
- Endpoint: [epid | attributes]
  - sendReqs : application messages to send
  - rsrcs : resrouce map from resource names to values
  - w4Ack: confirmable messages waiting for an ACK
  - rspRcd, rspSent -- responses received/sent
  - parameters controlling ACK wait time, sending delay ...
- Semantics -- rewrite rules
  - rcvMsg -- dispatches according to message features
  - sendMsg -- from sendReqs
  - replayMsg -- if w4ack timesout

CoAP vulnerabilities - what can go wrong




```
dev0          eve          dev1
----          ---          lock
 o -PUTNDU-> @
 o -PUTNSO->  ------->      o
 o <-------  <-2.01--       o
 o -PUTNDL-> ------->       o
 o <-------  <-2.01--       o
             o -PUTNDU->    o
             X <-2.04-      o
```

- A signal from the client starts a process that needs the door to be unlocked.
- The unlock message from the client is blocked process may have unexpected effects even if the server claims the signal has been received.
- The client locks the door, then eve releases the blocked unlock message and blocks the response.

CoAP vulnerabilities - what can go wrong


```
dev0              eve         dev1
---               ---  oven/tt  room/t2
 o    ----GETN(o)--->     o
 |                 @ <-t1-    o
 o -GETN(r) -> X          |
 o <-t1-        o          |
```

- Client requests temperature from oven and from room.
- The oven reply is delayed and the room request is blocked.
- The client may interpret the delayed oven repy as the room reply and suspect fire.

Attack models

- *Passive attacker* can listen, transmit messages it constructs, receive messages
- *Active attacker* can listen, drop, delay, replay (with modification)
- The above examples are active attacks
- *Reactive attacker* can listen, copy and replay with possibly modified sender or receiver; can not change the original messages in the network
- The above examples can be transformed to reactive attacks

CoAP Dialect

```
CoAP    D       D        CoAP
  | -M>| -D(M)> | -U(D(M))>|
  |    |        |          | U(D(M)) = M
  |<M1-|<D(M1)- |<M1-      |
  |    |        |          |
```

-M> CoAP client sends M

-D(M)> dialect layer obfuscates M

U(D(M))> server diaect layer deobfuscates

- Moving target; the functions D,U have parameters that change periodically (possibly every message) and unpredictabley
- Synchronization--how the receiver determines which parameters to use--is a challenge:
  - messages may be dropped or delayed
  - messages may arrive out of order
- Thus simply counting or time based synchronization doesn't work

CoAP Dialect Functions

The CoAP dialect specification is parameterized by 3 functions

- *g(seed,len,ix)* : *String* - encapsulates a generator of (pseudo) randomness.
    - *seed* is the generator seed
    - *len* —output length, *ix*— index into the generated sequence
- *f1(bits,content,ix) : DCBits* - obfuscates the content
    - bits is a source of randomness
    - content is the message content to be obfuscated
- *f2(bits,(dcbits,ix)) : Content x Nat* — the de-obfuscator
    - *dcbits* — the obfuscated content
- Each communicating pair in the network shares a unique secret.
- Each CoAp instance keeps a message sent counter for each partner.

CoAP Dialect Function Requirements

1. *f2* recovers the original content encoded by *f1*

   f2(grand, {f1(grand,content,ix)},ix)
= *{content},ix}*

2. if the encoded content or index are modified, decoding will fail.

The choice of *g,f1,f2* involve tradnng use of device resources against degree of harm an attack could do.

Obfuscation examples include bit permutation, xor, ...

Dialect transformation

We use the Russian dolls model (nested configurations) to represent a dialected CoAP system:

D : [epid | devattrs] >>
[epid | conf([epid | devattrs] localnet) dialectattrs]

U: [epid | conf([epid | devattrs] localnet) dialectattrs]
>> [epid | devattrs]

Dialect receive/send rules apply the dialect functions.

CoAP rules apply to the nested configuration withoug change

Dialect Properties

Assume CoAP systems running on an unreliable network

Non interference:

In absence of attacks a CoAP system, *ISys(mtC)*,and its dialected version, *D(ISys(mtC))*, are stuttering bisimilar

D(ISys(mtC)) <~~> *ISys(mtC)*

Protection:

A dialected CoAP system, *D(ISys(C))*, with attacker of capability *C* is stuttering bisimilar to the same systme in the absence of attack, *ISys(mtC)*

D(ISys(C)) <~~> *ISys(mtC)*

if *C* is any combination of drop or delay(n), (network attacker), or *C* is r*eplay(n)* with possible diversion (edit source and/or destination).

D turns attack into drop

Application Layer

The application layer adds an attribute to the CoAP endpoint state

[epid | atts aconf(abnds,arules)]

abnds -- the application local knowledge base (AKB)

arules -- rules the specify the application behavior

The CoAP layer passes each incoment message to the application layer after normal processing

Am application rule has a pattern to decide if the rule apples to an incomenting message and a condition to determine if its actions are enabled by the specific message in the current application state.

Rule actions can

send a message

update the local KB, or update the CoAP level resource map

```
app coap   coap app
 | ->|       |   |
 |   |-req-> |   |   req
 |   |       |K->| effects
 |   |<resp- |   |
 | <-|       |   |
```

Movable Bridge Example

bs -> bctl : boat here -- a boat wants to pass

bctl becomes working

bctl -> bs : received

bctl -> ga : GateCl -- clear traffic and close

ga -> bctl : success -- gate is closed

bctl -> br : BrigeOp

br -> bctl : sucess -- bridge is open

bctl -> bs : BSPass -- boat can pass

bs -> bctl : success -- boat has passed

bctl -> br : BridgeCl

br -> bctl : success -- bridge is closed

bctl -> ga : GateOp

ga -> bctl : success -- gate is open

bctl becomes idle

Primitives for constructing invariants

hasV(conf,epid,path,val) is true in configuration, *conf*, if the device with identifier *epid* has the binding *rb(path,val)* in its resource map.

hasAV(conf,epid,path,val) is true in configuration, *conf*, if the device with identifier *epid* has the binding *rb(path,val)* in its application layer KB.

isV(conf,ctl,epid,aid,path, val) is true in configuration, *conf*, if *hasV(conf,epid,path,val)* and the device with identifier *ctl* knows this (has received a response to its request to set a value.

becomeV(conf,ctl,epid,aid,path,val) is true in configuration, *conf*, if *hasV(conf,epid,path,val)* holds and the device with identifier *ctl* is waiting for a response confirming this assignment.

Symbolic search for attacks

To make search for attacks more efficient, we use an attack pattern and let the search mechanism generate all attacks applicable for each message transmitted.

The *mcX(n)* attack pattern matches any request message, *m(dst,src,c(path))*. The possible attacks replay the message or variants *m(dst1,src1,c(path)* where the endpoint *dst1* has a binding for path.

If *c* is a GET request, a capabiliy to make a copy of the response *m(src1,dst1,c1)* to be sent to *src* from *tgt*, otherwise the client will ignore the response.

Movable Bridge attacks

Invariant	nRnd	mcX	msg
bclIdleInv	1	20	GateCl,BridgeOp
	2	40	GateCl,BridgeOp
brNClInv	1	20	BridgeOp
	2	20	BridgeOp
gateNClInv	1	20	BridgeOp
	2	20	BridgeOp
boatPassInv	1	20-40	none
	2	20	BridgeCl,GateOp

Summary of bridge application attacks. The column nRnd is the number of rounds, mcX is the delay argument to the mcX attack capability, and msg is the message identifier of the attacked message.

Conclusion

Described the CoAP messaging protocol and illustrated different attack models

Presented a CoAP dialect wrapper and its bisimulation properties

Introduced an application layer with basic functions for expressing invariant, and symbolic search for attack. Illustrated with scenarios

Some future directions:

- Study multi message attacks
- Search for attack on more complex properties
- Methods to analyze specific dialect functions
- Methods to mitigate the network as attacker

Related Work and References

The CoAP standard -- basis for executable specification.
Rfc 7252: The constrained application protocol (CoAP)
Z. Shelby, K. Hartke, and C. Bormann. June 2014.

CoAP vulnerabilities -- basis for a case study.
Attacks on the constrained application protocol (CoAP)
J. P. Mattsson, J. Fornehed, G. Selander, F. Palombini, and C. Amsuss. Internet Draft, Network working group, 2023.

Formal framework developed in companion project.
Protocol dialects as formal patterns. D. Gala´n, V. Garc´ıa, S. Escobar, C. Meadows, and J. Meseguer. ESORICS 2023

Dialects for CoAP-lik Messaging Protocols, Carolyn Talcott arXiv:2405.13295v1, May 2024 (this talk)

The Maude specification and case studies can be found at ttps://github.com/SRI-CSL/VCPublic in the folder CoAPDialect.