TinyContainer
an abstraction and resource control layer for containers on RIOT

Chrystel Gaber & Samuel Legouix
RIOT Summit 2023

https://github.com/TinyPART/RIOT/tree/tinycontainer
TinyContainer in a nutshell

Think of containers on IoT devices as small, fortified self-contained islands in a vast digital archipelago. Each of them houses unique resources within its secure borders.

TinyContainer is a solution that builds bridges to connect these islands and enforces only authorized passage on them, enabling the controlled exchange of resources.
TinyPART Consortium

- German-French project in the joint Call on Cybersecurity
- 5 partners

- Use Case: Federated learning (FUB–PhySec)
- 4 enablers
  - PIP-MPU (Orange & UnivLille)
  - RIOT (INRIA)
  - TinyContainer (Orange & Inria & FUB)
  - Differential Privacy (FUB)

https://tinypart.github.io/TinyPART/
TinyPART: Industry needs

- **EU GDPR** requires to not expose user private data
- **EU Cybersecurity Act** requires certification of IoT devices
- **EU CyberResilience Act** reinforces Supply Chain Security
- **Certification level** depends on the risk of a specific usage context (e.g. risks on privacy or injuries)
- **Contexts evolve** and may require adaptation to new levels of risk (e.g. fire)

**Personalize customer relationship**

- Personalization of services make the customer feel unique and is a customer engagement driver,
- In post-Covid world, customers expectations blend human and digital,
- Privacy becomes a selling point.

**Optimize Time To Market**

- Development and security require time and expertise

**Comply with regulatory requirements**

- EU GDPR requires to not expose user private data
- EU Cybersecurity Act requires certification of IoT devices
- EU CyberResilience Act reinforces Supply Chain Security
- Certification level depends on the risk of a specific usage context (e.g. risks on privacy or injuries)
- Contexts evolve and may require adaptation to new levels of risk (e.g. fire)
# TinyPART: Our goals

## Investigate

- containers to *adapt* IoT logic to context
- capabilities to *isolate* untrusted IoT logic from multiple parties

## Strategy

- secure & private-by-design architecture
- open-source and modular development bricks
How can we manage isolation in a multi-party constrained device?
TinyPART: Context

Roles

- Service Providers
- End Users
- Device Maintainer
- Container Management Provider
- Device Owners

How can we manage isolation in a multi-party constrained device?
Target: Low-power IoT

IoT Class 2 *TinyPART target*

Currently out of reach

Available PoC without security
nrf52832 - 512 kB flash/64kB RAM

Target PoC with security
nrf52840 – 1MB flash/256kB RAM
TinyContainer

- Untrusted

API

- init()
- loadcontainer()
- startcontainer()
- stopcontainer()
- iscontainerrunning()
1) Container lifecycle management
1) Container lifecycle management

2) Runtime Abstraction
1) Container lifecycle management
2) Runtime Abstraction
3) Resources access management
1) Container lifecycle management
2) Runtime Abstraction
3) Resources access management
4) Multi-container support
Using TinyContainer
Container state machine

Provisionning
- LOADING
- INSTALL

Running
- STARTUP → ON_START → READY
- CONTINUE
- ON_LOOP
- SYSCALL

Stopping
- ON_STOP
- ON_FINALIZE

Container created
Create a container

1) Provision container
2) Create thread & container agent
3) Load & Start Container
During the container life

- **onstart()**
- **onstop()**
- **onloop()**
Periodic handover

heartbeat()
Ressources exposed to containers

Ressources are syscalls or endpoints
- Local
- Container
- Remote

Service: Manage endpoints
- Open
- Read
- Write
- Close

Firewall: Authorize calls
Endpoints & access rights management
What is a container?

Container

Code

Data

Metadata
### Metadata

<table>
<thead>
<tr>
<th>container</th>
<th>type</th>
<th>syscall_mask</th>
</tr>
</thead>
</table>

*syscall_mask => rights to native functions*
Metadata

container
  type
  syscall_mask \textsubscript{CWT}

endpoints
  endpoint_0
  endpoint_1
  endpoint_n

syscall_mask \Rightarrow \text{rights to native functions}

container
  container_id
  ...

endpoint_id
  peer_type
  ...
  ...
  authorization_code \textsubscript{CWT}

local
  bus_type
  ...

remote
  protocole_type
  ...

spi
  ...

i2c
  ...

gpio
  ...

https
  ...

dtls
  ...

authorization_code = Header
Payload =
  endpoint_id
  peer_type
  ...

\text{can be provided by a 3rd party}
Metadata

container
- container
- type
- syscall_mask $^{\text{CWT}}$

endpoints
- endpoint_0
- endpoint_1
- endpoint_n

security
- metadata_checksum
- data_checksum
- code_checksum
- container_lifetime
- vendor_secu

syscall_mask $\Rightarrow$ rights to native functions

container
- container_id
- ...

endpoint_id
- peer_type
- ...
- ...
- authorization_code $^{\text{CWT}}$

local
- bus_type
- ...

remote
- protocole_type
- ...

spi
- ...

i2c
- ...

gpio
- ...

https
- ...

dtls
- ...

authorization_code = Header Payload=

endpoint_id
- peer_type
- ...

can be provided by a 3rd party
Initialize container rights

1) Provision container

2) Hand over to Firewall

3) Configure rights
   a) Verify Checksums & tokens
   b) Assign ID to container
   c) Fill in container table
Next steps

- Performance & memory usage
- Root of Trust
- Integration with Pip-MPU
Food for discussion

- Feedbacks & use cases
- VM/interpreter syscall interface for bindings
- Towards an application firewall?

https://forum.riot-os.org/t/vm-interpreter-syscall-interface-for-bindings/3902
Live:
TinyContainer repo & helloworld

Recorded:
MyCoach use case
Demo: MyCoach

End User

Device Owner

Service Provider

Container Management Provider
Demo: MyCoach

Target Result

Results:
- Should press -> Was pressed: 24%
- Should release -> Was released: 45%
- Should press -> Was pressed: 25%
- Should release -> Was pressed: 5%
Demo: MyCoach

Before provisionning

After provisionning
Thank You

https://github.com/TinyPART/RIOT/tree/tinycontainer