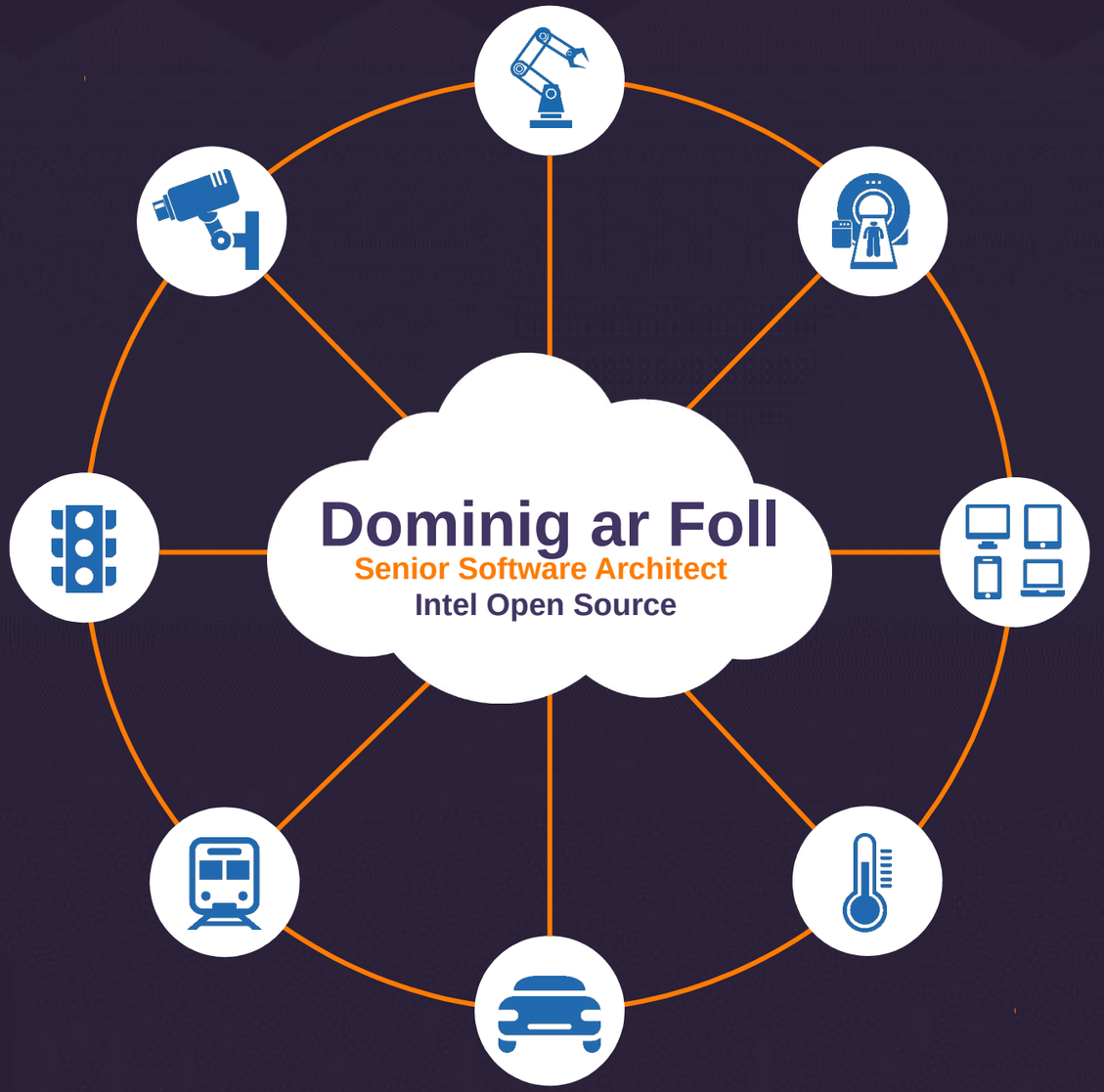


FOSDEM 2016, Bruxelles, Be dominig.arfoll@fridu.net

Odds are against us

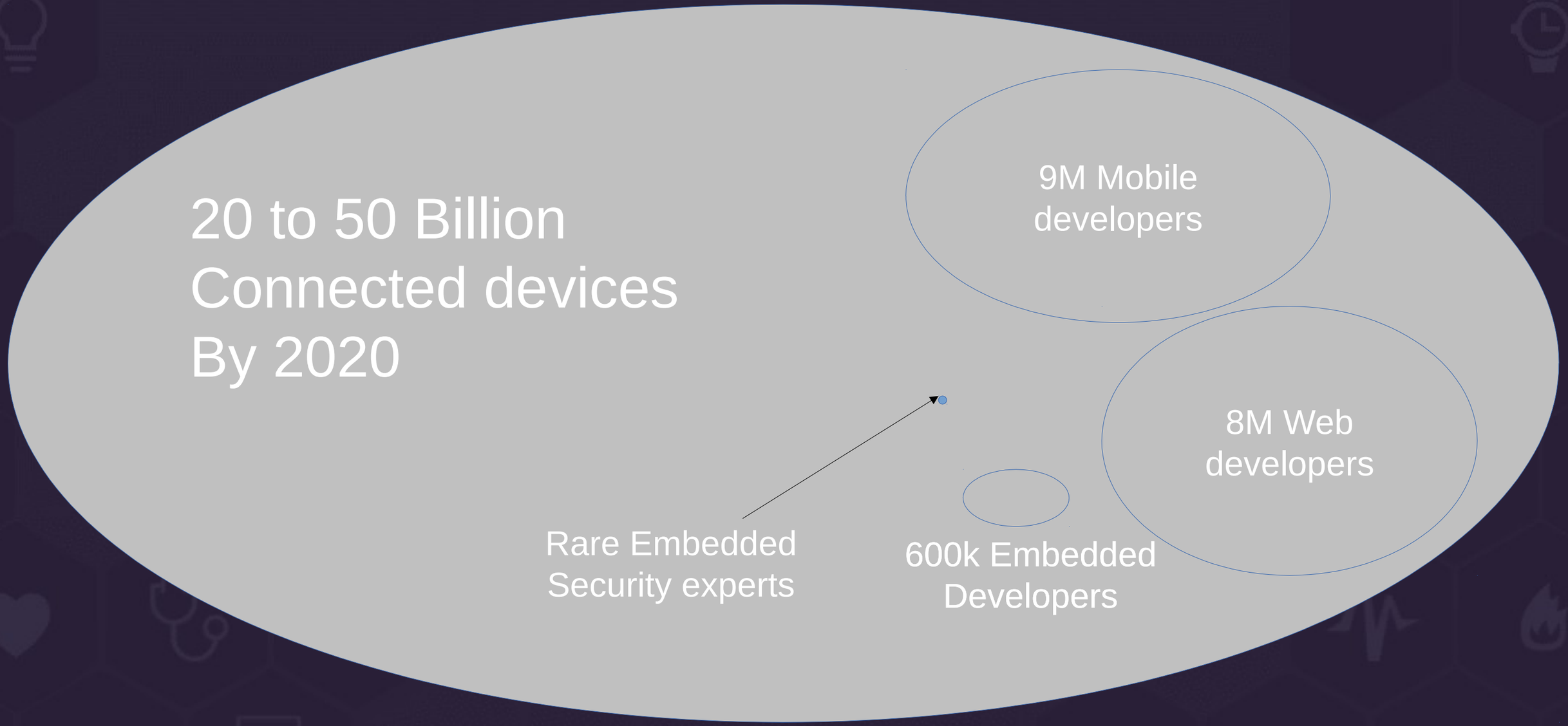
Developer

Fix all possible weaknesses
Deactivate possible users errors
LTS assumed for free

Back Hat

Only need one security hole
Can be help by careless users
Good long term business opportunities
Good international network

We cannot rely on “experts”

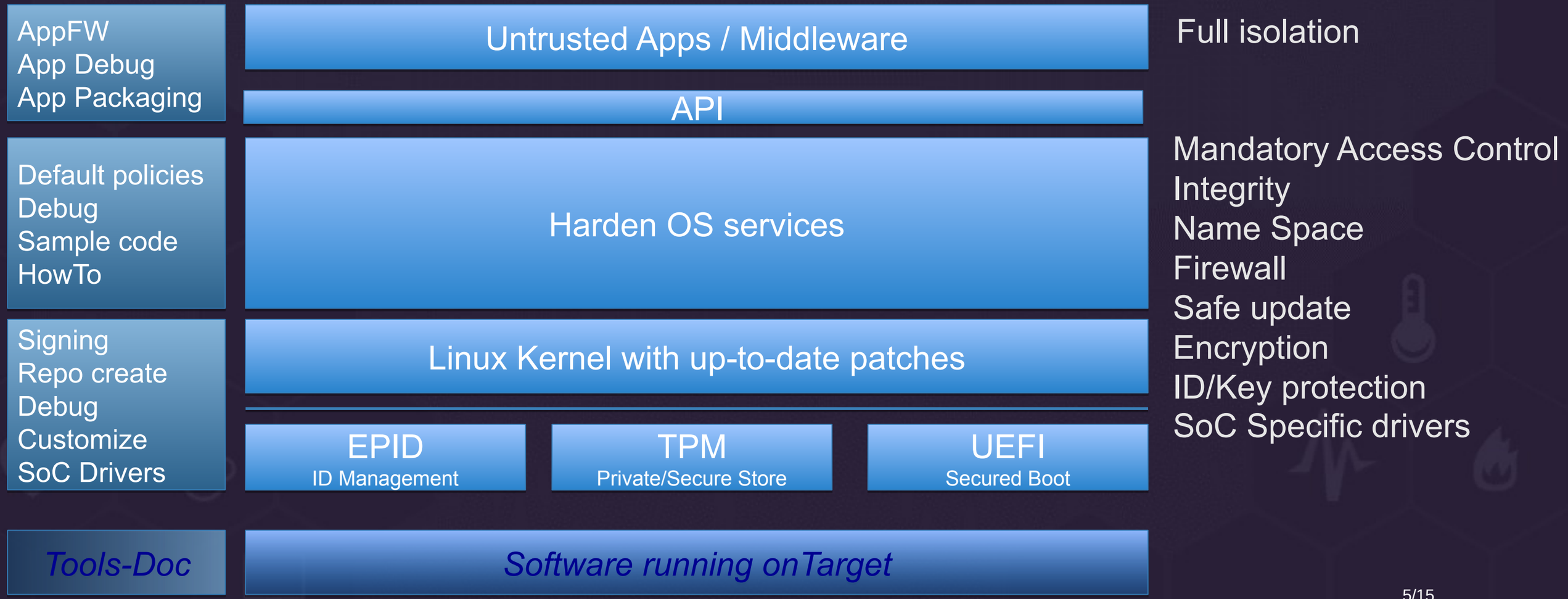


Back to the fundamentals

Minimise surface of attack
Control the code which is run
Provide a bullet proof update model
Track security patches in days rather than weeks
Use HW security helpers when available
Limit lateral movement in the system
Develop and QA with security turned on
Provide tools to Dev to enable secure development
Note that there is no miracle solution, it's will be hard work.

Security cannot be added after the fact

Designed for Security



Which code I run

Trusted boot UEFI is your friend

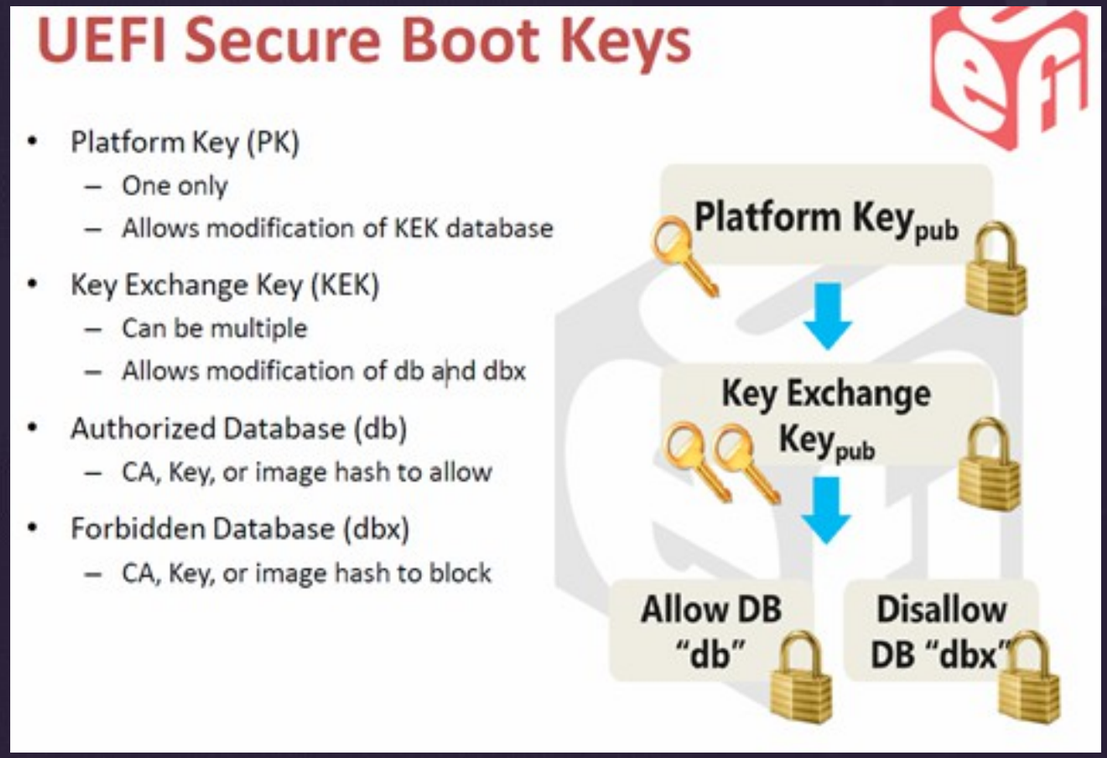
- Keys can be customised for small series
- API are well defined
- Supported on many HW.

Integrity

- Protects all critical files
- Optionally impose external signing

Update

- Only signed update
- Secured update on compromised device
- Factory reset built in
- Do not let back door open via containers
- Strict control on custom drivers



https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface
http://kroah.com/log/blog/2013/09/02/booting-a-self-signed-linux-kernel/
https://msdn.microsoft.com/en-us/library/windows/hardware/hh973604.aspx
http://sourceforge.net/p/linux-ima/wiki/Home/

Know who/what you trust

Trust code

- Keys in initial boot loader
- Signing images
- Signing update
- Simplified development process
- EPID is your friend

Track your App

- Signing App developed externally
- Dedifferentiate platform, partner, external developers

Track App call in the system

- MAC for local Apps
- Oauth 2.0 / OpenID / SAML for remote App

https://en.wikipedia.org/wiki/OAuth

Bullet proof update and ID

Update is the only possible correction

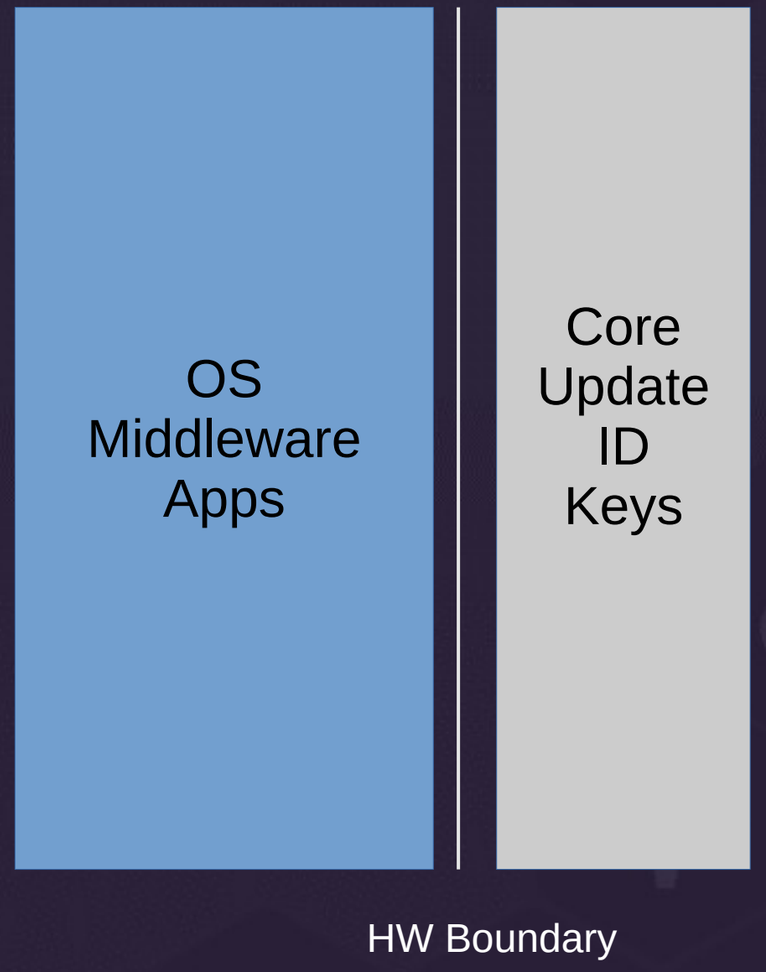
- Must run safely on compromised device
- Cannot assume a know starting point

Compromised ID / keys has no return

- Per device unique ID
- Per device symmetric keys
- Use HW ID protection (e.g. EPID)

Non reproducibility

- Breaking in one device cannot be extended
- Development I/O are disabled
- Root password is unique
- Password cannot be easily recalculated



Isolate services

Run services with UID<>0 SystemD is your friend

- Create dedicated UID per service
- Use MAC and DAC to minimise open Access

Drop privileges

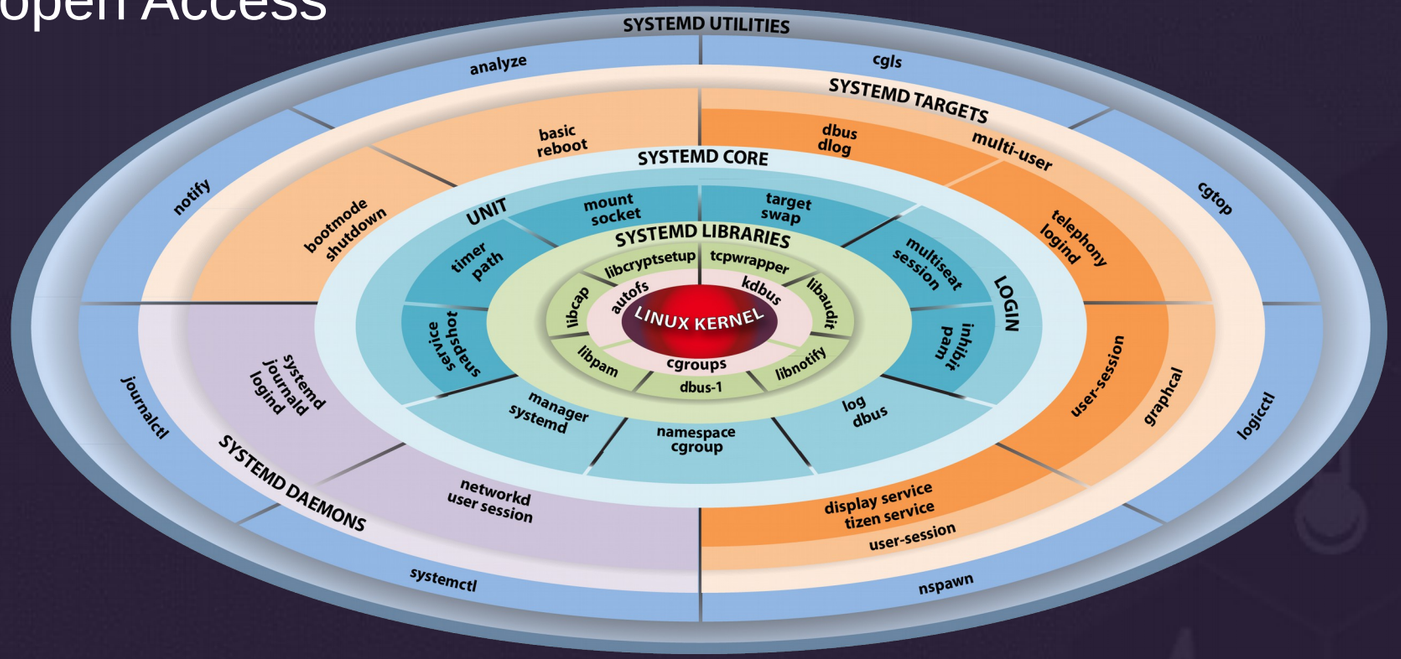
- Posix privileges
- MAC privileges

C-goups

- Reduce offending power
- RAM/CPU/IO

Name Space

- Limit access to private data
- Limit access to connectivity



https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt
https://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.2/capfaq-0.2.txt
http://man7.org/linux/man-pages/man7/namespaces.7.html
https://en.wikipedia.org/wiki/Mandatory_access_control
https://en.wikipedia.org/wiki/Discretionary_access_control

Isolate Apps

Apps use the OS, should not define it

- Apps cannot change the OS behaviour
- Apps privileges are limited

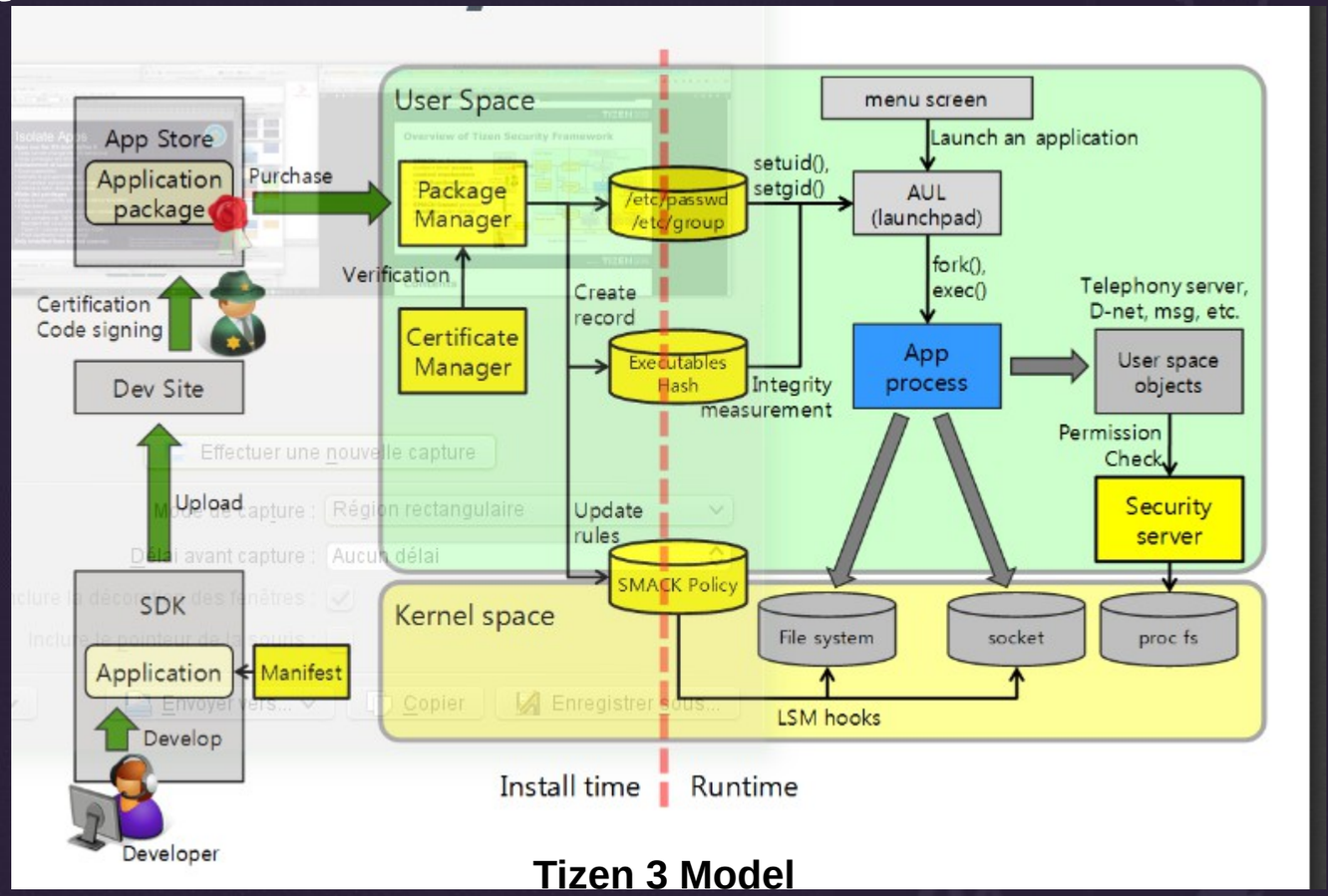
Containment at launch

- Drop capabilities
- Activate G-groups limitation
- Limit system access via name spaces
- Enforce a MAC unique App tracer

White list privileges

- If not explicitly allowed >stickly forbidden
- Enforcement
  - Simpler with API and MAC combination
  - Too complex via 100% MAC
  - standard SE Linux policy >120kl
  - Tizen 2.1 phone smack policy >30kl
  - Poor performer via Seccomp

Only installed from trusted sources



Tizen 3 Model

https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt
http://selinuxproject.org/page/NB_PolicyType
https://wiki.tizen.org/wiki/Security:SmackThreeDomainModel

Container "A mixed blessing"

Easy to use

- Detach the App from the platform
- Integrated App management
- Well known

Not very secure

- Unreliable introspection
- MAC has no power on the inside of a container
- Updating the platform does not update the middleware
- Beside the Kernel each App provide its own version of the OS
- Each App restart requires a full passing of credential
- RAM and Flash footprint are uncontrollable
- Far more secured with Clear Container but not applicable to low end SoC.

Only I/O via network

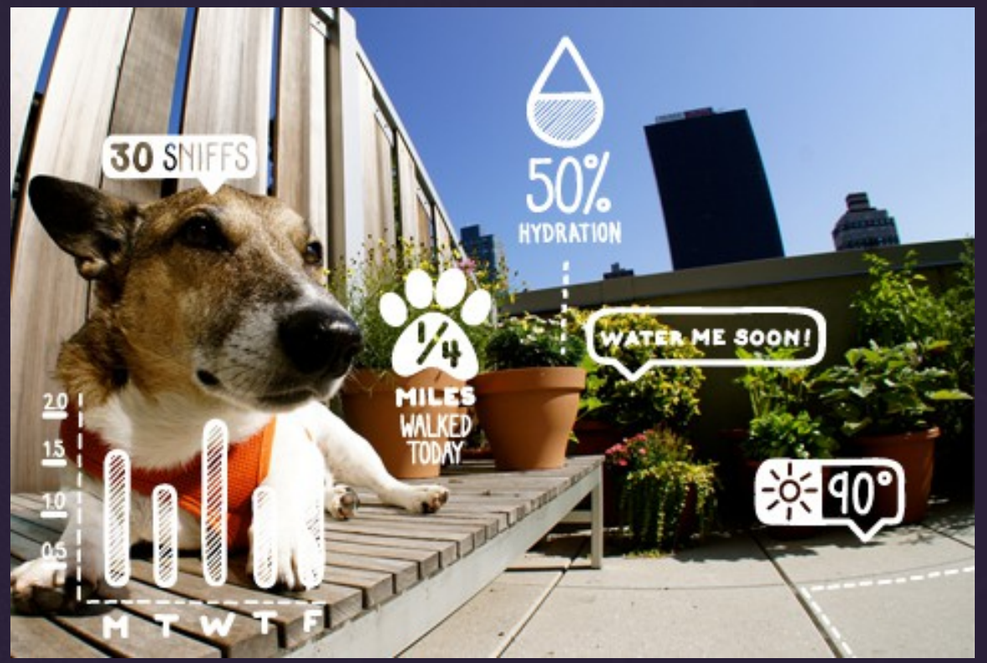
- Well equipped for Rest API
- All other I/O requires driver level access or bespoke framework.

https://www.opencontainers.org/
https://lwn.net/Articles/644675/

No relying on end user

End user as an IT manager – a No No

- App or Browser as UI
- Dead simple but secured
- Unified on various generation of devices
- Possible actions
  * reboot/update
  * return to the shop



Provider managed

- user still need to customise
- multiple EMS stacks (TR-069, M2M, …)
- devices behind NAT

Development process

Filter code in

- Code reviews
- Licence check
- Static analysis
- Silent Dependency

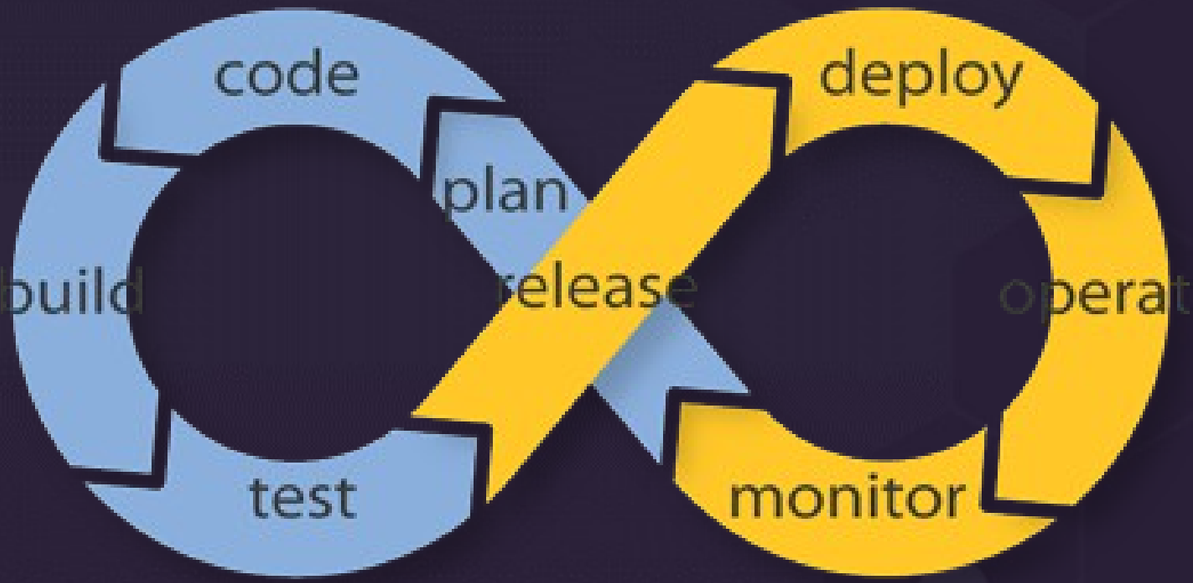
Auto test

- Compile/build test
- Automatic test
- Run test on secure image

Develop with Security on

- Provide Devel image with configurable security
- Enforce full respect of Service and App isolation

Do not accept any temporary "security disable"



https://www.opencontainers.org/


Questions

Security Check list

Control which code you run

- Secure boot
- Integrity
- Secure update

Isolate services

- Drop root when possible
- Drop privileges

Isolate Apps

- Apps are not the OS
- Enforce – restrict access to standard API

Identity

- Enforce identity unicity
- Use available HW protection

Encryption

- Network traffic
- Local storage

Control image creation

- No debug tool in production
- No default root password
- No unrequired open port

Continuous integration

- Automate static analysis
- QA on secured image

Help developer

- Integrate security in Devel image
- Provide clear guide line
- Isolate Apps from OS
- Focus on standardised Middleware