Konsulko Group

ICS

Software Update Mechanisms:

Selecting the Best Solution for Your Embedded Linux Device

About ICS

Delivering Smart Devices for a Connected World

- Founded in 1987
- Largest source of independent Qt expertise in North America
- Trusted Qt Service Partner since 2002
- Exclusive Open Enrollment Training Partner in North America
- Provides integrated custom software development and user experience (UX) design
- Embedded, touchscreen, mobile and desktop applications
- HQ in Waltham, MA with offices in California, Canada, Europe

Qt Service Partner

Boston UX

- Part of the ICS family, focusing on UX design
- Designs intuitive touchscreen interfaces for high-impact embedded and connected medical, industrial and consumer devices

BOSTON UX

A PREMIER EMBEDDED LINUX
ENGINEERING SERVICES COMPANY

Konsulko Group

A PAST, PRESENT, AND FUTURE
IN OPEN SOURCE SOFTWARE

Presenters

Jeff Tranter

Qt Consulting Manager

ICS

Leon Anavi

Senior Software Engineer

Konsulko Group

Software Update Mechanisms

- Many things to consider (disk space, network bandwidth, security, apps, etc)
- Build systems and board support packages (BSP)
- Integration of the update mechanisms
- Back-end for managing fleets of devices

Build Frameworks for Embedded Linux Distributions

Popular open source build systems for custom embedded Linux distributions:

yocto PROJECT

openembedded

PTX DIST

OpenWrt WIRELESS FREEDOM

debian

Core

Open Build Service

Which one do you use?

The Yocto Project

- Open source collaborative project of the Linux foundation for creating custom Linux-based systems for embedded devices using the OpenEmbedded Build System
- OpenEmbedded Build System includes BitBake and OpenEmbedded Core
- Poky is a reference distribution of the Yocto Project provided as metadata, without binary files, to bootstrap your own distribution for embedded devices
- Bi-annual release cycle
- Long term support (LTS) release covering two-year period

Common Embedded Linux Update Strategies

- A/B updates (dual redundant scheme)
- Delta updates
- Container-based updates
- Combined strategies

A/B Upgrades

- Dual A/B identical rootfs partitions
- Data partition for storing any persistent data which is left unchanged during the update process
- Typically a client application runs on the embedded device and periodically connects to a server to check for updates
- If a new software update is available, the client downloads and installs it on the other partition
- Fallback in case of update failure

Delta Updates

- Only the binary delta between the difference is sent to the embedded device
- Works in a Git-like model for filesystem trees
- Saves storage space and connection bandwidth
- Rollback of the system to a previous state

A/B vs Delta Updates

Update strategy	Storage space	Update size	Rollback to a previous stage	Fallback to a back-up image on a separate partition
A/B Updates	Large	Large	Yes	Yes
Delta Updates	Small	Small	Yes	No

Container-based Updates

- Container technology has changed the way application developers interact with the cloud and some of the good practices are nowadays applied to the development workflow for embedded Linux devices and Internet of Things
- Containers make applications faster to deploy, easier to update and more secure through isolation
- Yocto/OE layer meta-virtualization provides support for building Xen, KVM, Libvirt, docker and associated packages necessary for constructing OE-based virtualized solutions

Combined Strategies

Multiple combinations exist for more complicated use cases, for example:

- A/B updates with delta updates
- Containers with A/B updates
- Containers with delta updates
- Containers with delta and A/B updates
- Other

Popular open source solutions

- Mender
- RAUC
- SWUpdate
- Swupd
- UpdateHub
- Balena
- Pantacor
- Eclipse Kanto
- Snap
- libostree (OSTree)
- Aktualizr
- Aktualizr-lite
- QtOTA
- Torizon
- FullMetalUpdate
- Rpm-ostree (used in Project Atomic)

Mender

- Available as a free open source or paid commercial/enterprise plans
- **A/B update scheme** for open source and all plans as well as **delta updates** for professional and enterprise plans
- Back-end services (Hosted Mender)
- Written in Go, Python, JavaScript
- Yocto/OE integration through meta-mender and extra BSP layers: https://github.com/mendersoftware/meta-mender
- Support for Raspberry Pi, BeagleBone, x86-64, Rockchip, Allwinner, NXP, etc.
- Source code in GitHub under Apache 2.0

How to do A/B Upgrade with Mender?

Steps to install Mender A/B update on an embedded device:

- Apply update
- Reboot
- On the first boot after a successful update, though the Mender client a commit must be performed to accept the update (otherwise the system will roll-back on next reboot)

Mender Client

Mender A/B updates supports two client modes:

- Managed (default) - client running as a daemon polls the server for updates
- Standalone - updates are triggered locally which is suitable for physical media or any network update in pull mode

To use standalone client mode in Yocto/OpenEmbedded set:

SYSTEMD_AUTO_ENABLE_pn-mender = "disable"

Mender Data Partition

- Mender creates a **/data** partition to store persistent data, preserved during Mender updates. Supports ext4, Btrfs and F2FS file systems.

- The Mender client on the embedded device uses **/data/mender** to preserve data and state across updates

- Variable **MENDER_DATA_PART_SIZE_MB** configures the size of the **/data** partition. By default it is **128MB**. If enabled, mender feature **mender-growfs-data** which relies on **systemd-growfs** tries to resize on first boot with the remaining free space

- It is possible to create an image for the data partition in advance with bitbake: **IMAGE_FSTYPES:append = " dataimg"**

RAUC

- A lightweight update client that runs on an Embedded Linux device and reliably controls the procedure of A/B upgrading the device with a new firmware revision
- Supports multiple update scenarios
- Provides tool for the build system to create, inspect and modify update bundles
- Uses X.509 cryptography to sign update bundles
- Compatible with the Yocto Project, PTXdist and Buildroot

RAUC

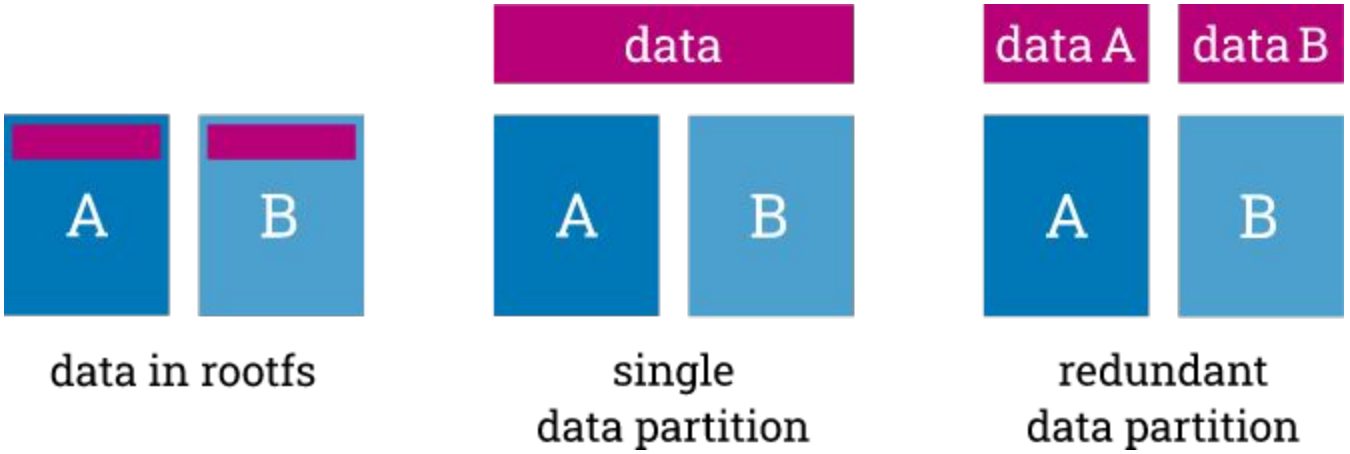
- Select an appropriate bootloader
- Enable SquashFS in the Linux kernel configurations
- ext4 root file system (RAUC does not have an ext2 / ext3 file type)
- Create specific partitions that matches the RAUC slots
- Configure Bootloader environment and create a script to switch RAUC slots
- Create a certificate and a keyring to RAUC’s system.conf

meta-rauc-community

- Yocto/OE layer with examples how to integrate RAUC on various machines
- Started in 2020
- Moved to the RAUC organization in GitHub in 2021
- Currently supports x86-64, QEMU, Raspberry Pi through meta-raspberrypi, Sunxi (Allwinner) devices through meta-sunxi, NVIDIA Jetson TX2 through meta-tegra
- https://github.com/rauc/meta-rauc-community/

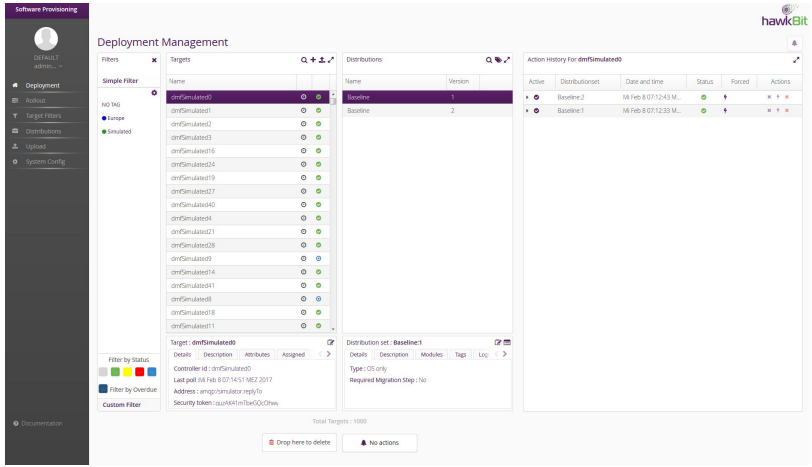
RAUC Data Partition

- Supports single and redundant data partitions
- For redundant data partitions the active rootfs slot has to mount the correct data partition dynamically, for example with a udev rule



One more thing: Eclipse hawkBit

- Domain independent back-end framework for rolling out software updates to constrained edge devices as well as more powerful controllers and gateways connected to IP based networking infrastructure
- Written in Java
- Available in GitHub under EPL-1.0 License
- Compatible with RAUC and SWUpdate
- https://www.eclipse.org/hawkbit/



libostree

- Libostree (previously known as OSTree), is a shared library and tools for distributing and deploying file system images as atomic upgrades of the whole file system tree
- Provides a suite of command line tools that combines a “git-like” model for committing, downloading and deploying bootable filesystem trees as well as bootloader configuration
- Effective with respect to disk space and connection bandwidth
- Supports rollback of the system to a previous state
- Used in many embedded Linux solutions: Fedora IoT, Torizon, Aktualizr, Aktualizr-lite and QtOTA

libostree

- libostree encourages systems to implement UsrMove https://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/
- libostree comes with optional dracut+systemd integration: https://ostreedev.github.io/ostree/adapting-existing/
- libostree only preserves **/var** across upgrades
- **/var** is empty by default and the operating system needs to dynamically create the targets of these at boot, for example with **systemd-tmpfiles** (if using systemd)

Combined Strategies with Containers

- Yocto/OE layer meta-virtualization provides support for building Xen, KVM, Libvirt, docker and associated packages necessary for constructing OE-based virtualized solutions. In Yocto/OE **virtualization** has to be added to the **DISTRO_FEATURES**:

DISTRO_FEATURES:append = " virtualization"

- For example adding Docker to the embedded Linux distribution is easy:

IMAGE_INSTALL:append = " docker-ce"

- There are use cases on powerful embedded devices where containers are combined with A/B updates of the base Linux distribution built with Yocto/OE, for example with RAUC or Mender

Conclusions

- There are numerous things to consider when choosing an upgrade strategy
- There are many reliable open source solutions and not worth developing another proprietary homegrown solution
- Combined update strategies such as A/B upgrades with containers for applications are increasingly popular nowadays
- The update process implementation depends on the build system and the bootloader
- Often real-world solutions require a persistent data partition which is left unchanged during the update process

Leveraging Hardware Support

HSM: Hardware Security Module.

TPM: Trusted Platform Module (also known as ISO/IEC 11889).

CAAM: Cryptographic Accelerator and Assurance Module (NXP i.MX6 processor).

CAAM (Cryptographic Accelerator and Assurance Module)

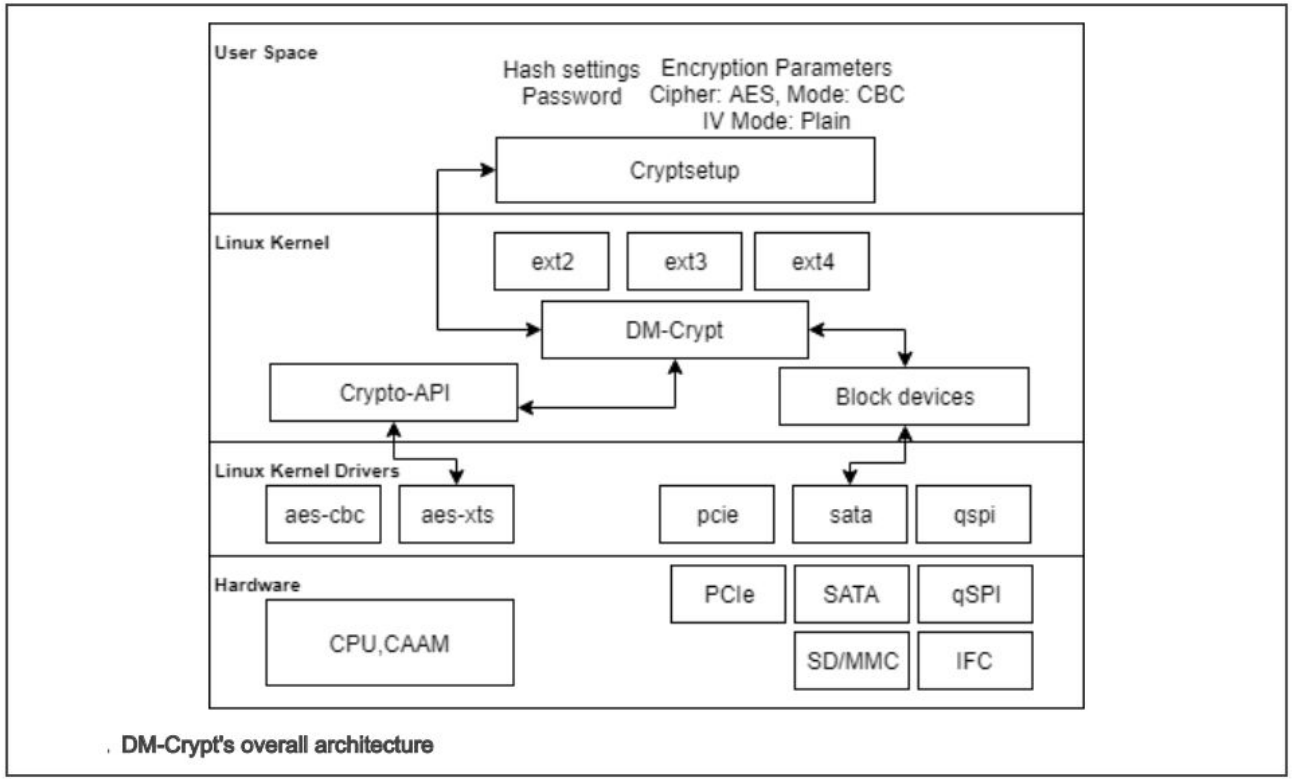
CAAM on the Freescale i.MX platform supports the following:

- Secure memory feature with hardware-enforced access control
- Cryptographic authentication:
  - Hashing algorithms: MD5, SHA-1, SHA-224, SHA-256
  - Message authentication codes (MAC): HMAC with all hashing algorithms, AES-CMAC, AES-XCBC-MAC
  - Auto padding
  - ICV checking
- Authenticated encryption algorithms:
  - AES-CCM (counter with CBC-MAC)

CAAM (Cryptographic Accelerator and Assurance Module)

- Symmetric key block ciphers:
  - AES (128-bit, 192-bit or 256-bit keys)
  - DES (64-bit keys, including key parity)
  - 3DES (128-bit or 192-bit keys, including key parity)
  - Cipher modes: ECB, CBC, CFB, OFB for all block ciphers, CTR for AES
- Symmetric key stream ciphers:
  - ArcFour (Alleged RC4 with 40 - 128 bit keys)
- Random-number generation:
  - Entropy is generated via an independent free-running ring oscillator.
  - For lower-power consumption, oscillator is off when not generating entropy.
  - NIST-compliant, pseudo random-number generator seeded using hardware-generated entropy.

CAAM (Cryptographic Accelerator and Assurance Module)



DM-Crypt's overall architecture

CAAM (Cryptographic Accelerator and Assurance Module)

Advantages:

- Hardware acceleration reduces CPU time and power usage.
- Validated algorithms.
- Secure method of storing keys.
- Transparent to application-level software when supported by third-party libraries.
- Built into processor/SOM.

Disadvantages:

- Hardware dependent.
- Operating system dependent.
- More complex to use directly.
- Can add to hardware cost if not part of SOM (e.g. HSM/TPM).

Using Microsoft Azure IoT to Host Software Updates from the Cloud

- Hosting network updates needs server side support.
- Can roll your own, but makes sense to look at commercial solutions if you need scalability, reliability, security, authentication, etc.
- Many possible solutions, but one option we have used is Microsoft Azure IOT.
- Provides libraries for clients that are cross-platform (scalable from MCUs to desktops) and multi-language (C, C++, Python).
- Often use "digital twin" architecture.
- We have used it for hosting software updates as well as a larger IoT support for managing devices like medical instruments to allow managing and monitoring a large fleet of instruments geographically distributed.

Using Microsoft Azure IoT to Host Software Updates from the Cloud

ICS Approach:

- Wrote a C++ wrapper library around Microsoft's Azure IoT client APIs to provide just the functions we need on the device.
- Makes it less dependent on Azure APIs, versions, etc.
- Avoids application developers needing to know details of Azure IoT.
- Also provides other features like authentication and key management.
- Have been able to use this wrapper library on two different projects so far.

Case Study - Software Updates for a Medical Instrument

Key Requirements:

- Ability to update application, operating system, and/or embedded firmware (e.g. MCUs).
- Encrypted and authenticated/signed updates.
- Desire to use similar scheme for updates of language translations and assays.
- Ability to rollback update on failure.
- Requirement to keep existing instrument data.
- Initially just support for updates from USB storage, but wanted to later extend to network updates.

Case Study - Software Updates for a Medical Instrument

Features of the design:

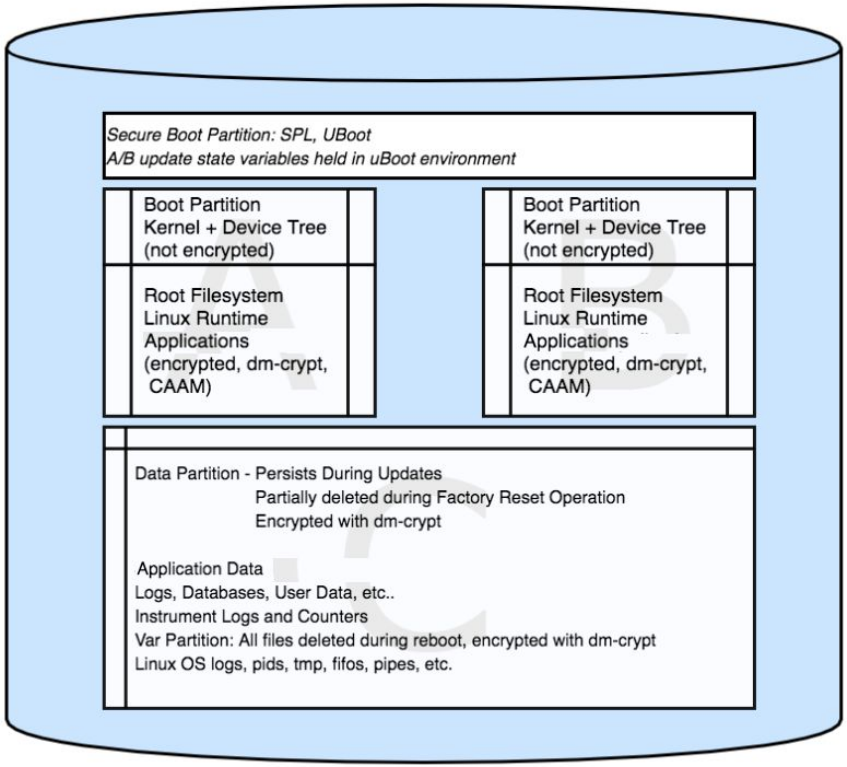
- A/B update approach.
- Components installed on the system (including software updates) use a manifest file which define the components of an update and have cryptographic hashes of all files. Manifests are signed and portions are encrypted. Overall updates are encrypted.
- Use certificates to generate and validate signing of updates.
- Data transferred between the instrument device and remote servers is encrypted.

Case Study - Software Updates for a Medical Instrument

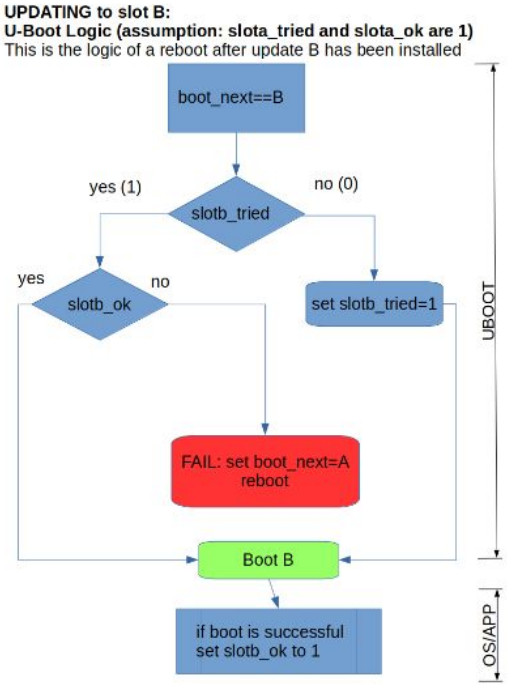
Features of the design (continued):

- TLS certificates verify the authenticity of the instrument that the servers communicate with.
- TLS certificates verify the authenticity of the remote servers that the instrument connects to.
- Leverage CAAM for encryption/decryption and key management.
- Data transferred over networking (internal and external to the device) encrypted using TLS with AES256 encryption using CAAM to manage keys and provide hardware acceleration of the encryption/decryption.

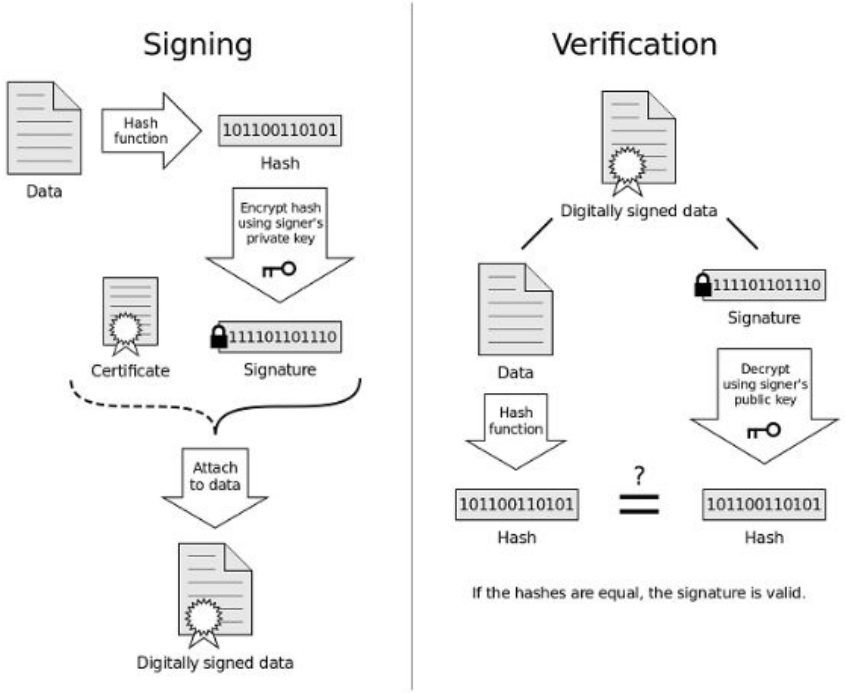
A/B Update Approach



A/B Update Approach



Signing and Verification



Secure Boot

Pros and Cons of a Roll Your Own Design

Pros:

- Fewer third-party components and dependencies.
- Custom solution addresses requirements of the particular project/device.
- No licensing costs or issues.
- Updates only contain the necessary components that need to be updated.

Cons:

- Was harder to expand system to support IoT than if it had been originally designed in.
- CAAM features are specific to i.MX6 platform.
- Challenging to get CAAM working.
- Design did not support updating the boot loader (but was not a requirement).

Other Lessons Learned

- Learning curve for CAAM
- Performance
- Security versus convenience

Any questions?

www.ics.com
www.konsulko.com