Shhh, stop sharing secrets!

A secure mindset for embedded development

Presented by: Andy Doan and Ricardo Salveti

EMBEDDED LINUX CONFERENCE

FOUNDRIES.IO

A classic “how it started” meme

The abstract:

I recently posted a support question to a major cloud provider to see how my embedded devices could securely connect with their private container registry. The answer - create a file with a username/password on each device. We can do better. We deserve better. Security for embedded products is complicated. There are multiple layers and dimensions so that "security" can't be turned into a simple one paragraph answer. However, if we step back to some first principles, we can create a mindset and approach for building secure embedded products.

This talk will cover ways to secure an embedded device. Hardware Security Modules(HSMs) and what they can do secure communication to the cloud will play a starring role. Topics like x509 PKI basics and ECIES encryption for securing data on devices will be covered as well. In the end, you'll be armed with some tricks to make your embedded product a little more secure.

Andy's Law of Security

Anything you don't want the world to see will eventually find its way to a public S3 bucket.

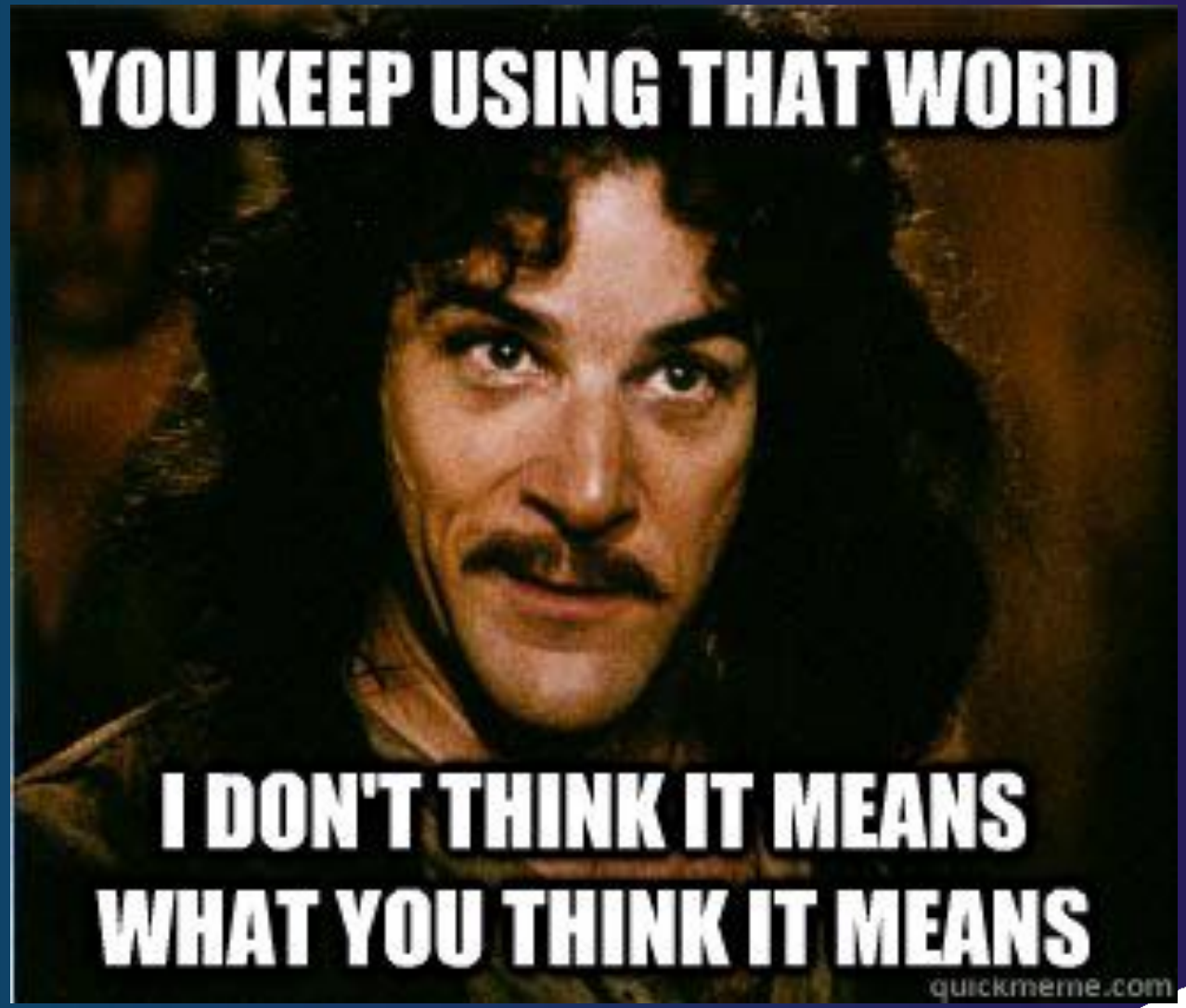
The “How hard could it be” mindset

- Or even worse, the Texan “here, hold my beer” mindset
- We are engineers
- We tend to think everything is easy

Security is really hard

FOUNDRIES.IO

“Security”



FOUNDRIES.IO

Building Blocks

- PKI - Public Key Infrastructure
  - x509 - ASN.1 is proof the universe hates you
- Encrypting stuff
  - Symmetric = Shared decryption key
  - Asymmetric = Only owner of private key can decrypt
- Signing stuff
  - Prove you posses the private key
- HSMs - Hardware Security Modules
  - There's not private key file to read

Combine the blocks together and you get chef's kiss embedded security

FOUNDRIES.IO

PKI High Level

- We have a root certificate authority (CA)
  - Private key and an x509 certificate signed by it
- Intermediate CAs
  - The root is “too big to fail” (see The Update Framework later)
  - The real work happens with these
- Certificate Signing Requests (CSR)
  - CSR includes information about an entity’s public key
  - Intermediate CA returns a “client certificate”
  - Because it has public key information you can do signing operations that prove you are in possession of the private key

PKI - Mutual TLS (mTLS)

- We have a “key infrastructure”
- mTLS allows clients and servers familiar with this PKI to trust one another:
  - The server advertises and “TLS certificate” and proof its owns the private key.
  - The client can then trust its talking to a legitimate service.
  - The server challenges the device for a certificate.
  - The device provides it’s certificate and proof it owns the private key.
- This is a great way to authenticate devices
  - All the big cloud providers have “API gateways” that support this.

FOUNDRIES.IO

Asymmetric Encryption High Level

- Why not symmetric encryption?
  - Some day your sensitive data will find its way to a public S3 bucket.
  - That same bucket will probably have the symmetric key

Asymmetric

- You take sensitive data and encrypt it with a device's public key.
- It's now a lot less worrisome to share the data with a 3rd party device-management system - they can't read it.
- Only the device can decrypt this content - do it in tmpfs/RAM!

ECIES side note

- Asymmetric encryption for RSA is well supported
- Asymmetric encryption for Elliptic curve is less so

And you you should be using EC!

Enter ECIES

- It's a standard but some of the implementations aren't interoperable.
- I've used the Ethereum (golang) version. Not interoperable with https://github.com/ecies/go

FOUNDRIES.IO

HSM High Level

- There’s no access to the private key.
- Math is amazing

credit: tenor.com

FOUNDRIES.IO

But, unfortunately life is not that "easy"

Security is only as strong as the weakest link

Using cryptography correctly and leveraging HSMs is a huge plus, but to tight things up it is also critically important to protect from local and physical threats, specially on IoT and embedded products.

"The only way to do security right is to have multiple layers of security" - Linus Torvalds

FOUNDRIES.IO

Security on Embedded Linux

FOUNDRIES.IO

Securing the platform and OS

- Secure boot
- Measured boot
- Remote attestation
- Disk encryption
- General hardening and surface attack reduction
  - Based on threat model analysis
  - Disabling things that are not required on production (e.g. JTAG)
- Abstract the final application in a contained environment
  - Designing Secure Containerized Applications for Embedded Linux Devices - Sergio Prado, Embedded Labworks
    - Friday - 12:00pm

FOUNDRIES.IO

Best practices with HSMs

While the keys are securely stored (hopefully) inside the element, access to the element also needs to be protected

- Encrypting the communication to the secure element
  - SCP03 (Secure Channel Protocol 3)
- Protect from direct access or reduce via permission/groups
- Reduce access surface to userspace via abstractions
  - Common interfaces such as PKCS#11
  - PARSEC (Platform AbstRaction for SECurity)
  - Abstract via Trusted Execution Environment

FOUNDRIES.IO

NXP SE05X abstraction via OP-TEE

- OP-TEE: open source Trusted Execution Environment (TEE) implementing the Arm TrustZone technology
  - Widely available on ARM SoCs
- Hardware root of trust extended to S05X
- Secure communication (SCP03) established by OP-TEE
  - Keys derived at runtime from the Hardware Unique Key
- Cryptographic operations done at hardware level
- Userspace abstraction via PKCS#11 library and TA

Embedded Recipes 2022 - Secure Elements in a Trusted Execution Environment - Jorge Ramirez

FOUNDRIES.IO

HSMs - unsolicited advice

- NXP has SE05X - works great for embedded products.
  - Can add EdgeLock 2GO to take things further
  - OP-TEE OS abstraction available upstream
- Softhsm - Anyone can prototype!

What about my RPi?

- Security or RPi - pick one

Things We Can Start Doing

- Use Mutual TLS (mTLS)
- Store mTLS key in the HSM
- If you know the device's public key, you can store/send it encrypted content that only it's HSM can decrypt
- Talk to 3rd party systems - the good ones have mTLS hooks
- Build software trustworthiness from the hardware root of trust
- Extend trust through all layers of the OS

FOUNDRIES.IO

Things We Can Stop Doing

- Copying sensitive data onto devices in clear-text.
- Copying AWS credentials to a device
  - https://aws.amazon.com/blogs/security/how-to-eliminate-the-need-for-hardcoded-aws-credentials-in-devices-by-using-the-aws-iot-credentials-provider/
  - https://foundries.io/insights/blog/aws-iot-jitp/
- If not AWS?
  - Use an mTLS protected APIs for Azure/GCP
    - Create an mTLS protected API
    - Have a lambda function return temporary credentials to each device
- Thinking about security as an afterthought
  - Make it part of the product development lifecycle
  - Have a way to update things when needed (from boot firmware to the OS)

FOUNDRIES.IO

Questions?

FOUNDRIES.IO