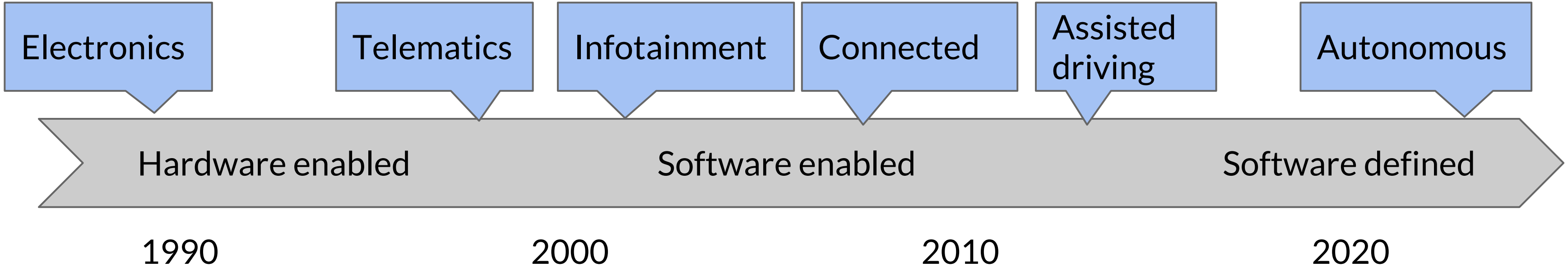
Securing the Connected Car

MENDER.io

Deploy Software Updates for Linux Devices

Eystein Stenberg
CTO
Mender.io

The software defined car



Electronics

Telematics

Infotainment

Connected

Assisted driving

Autonomous

Hardware enabled

Software enabled

Software defined

1990

2000

2010

2020

About me

- Eystein Stenberg
  - 7 years in systems security management
  - M. Sc., Computer Science, Cryptography
  - eystein@mender.io

- Mender.io
  - Over-the-air updater for Linux, Yocto Project
  - Open source (Apache License, v2)
  - Dual A/B rootfs layout (client)
  - Remote deployment management (server)
  - Under active development

Session overview

- Opportunities with the software defined car

- Anatomy of an attack: security risks of the connected car

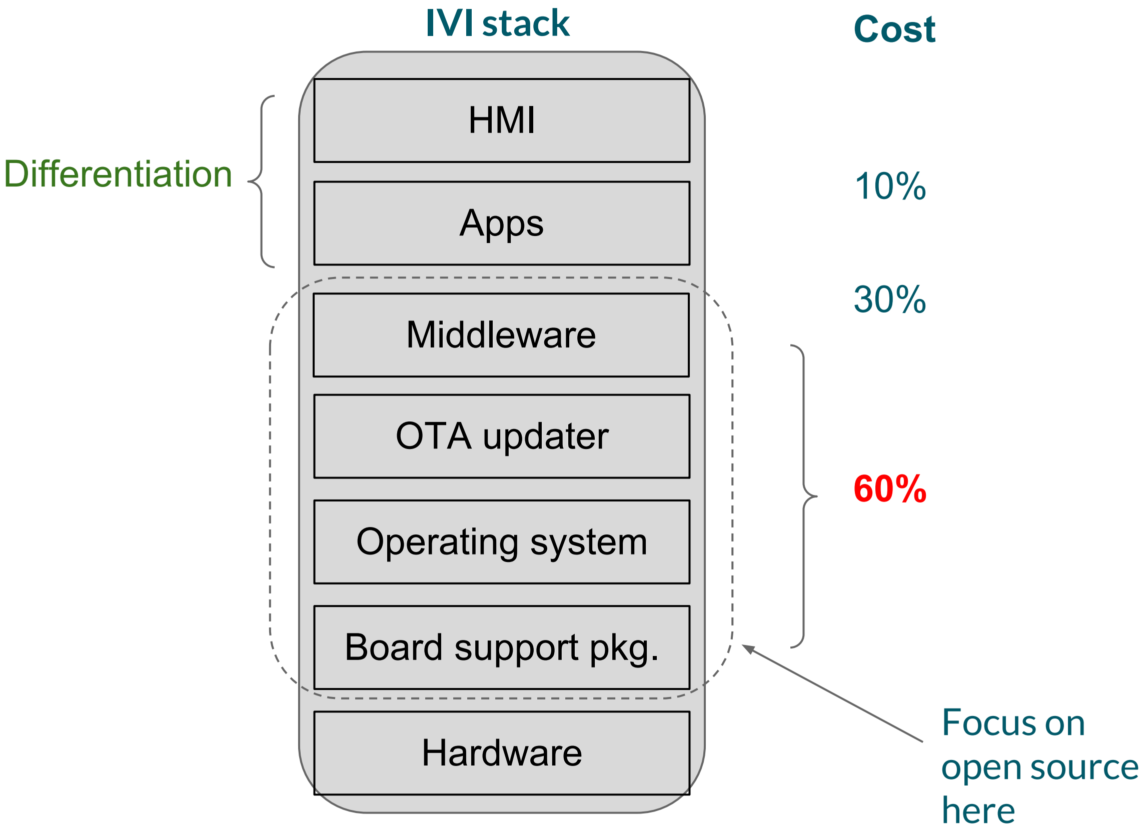
- The patching problem & solution designs

Software defined car: New revenue streams

- "Automakers could add up to **$27.1B annually** from services such as car sharing and more" - Navigant Research

- Tesla
    - An OTA update system allows for easy additional software purchases **after** buyers drive their cars off the lot
    - Semi-autonomous Autopilot feature allows current Model S owners to add the feature for $2,500 USD when they order the vehicle or they can pay $3,000 USD to upgrade later

Cost savings by using open source platforms



IVI stack

Cost

Differentiation

HMI

Apps

10%

30%

Middleware

OTA updater

Operating system

Board support pkg.

60%

Hardware

Focus on open source here

- Lower layers are *expensive* and provides *no differentiation*
- Use open source here to
  - Shorten time-to-market
  - Lower cost
  - Reallocate development to differentiating features

The software defined car requires OTA updates

- Increased software complexity requires more frequent improvements
- “**33% of current recalls** are for problems that could be fixed OTA” - ABI Research
- “OTA updates will **save carmakers $35B** in 2022” - IHS Automotive
- Fiat Chrysler hack (next up) required a **recall of 1.4 million vehicles** that could have been avoided with an OTA update

Jeep Cherokee hacked in July 2015

- Presented at Black Hat USA 2015
  - Charlie Miller
  - Chris Valasek
- Remote exploit giving full control of the car
- Clearly demonstrates physical safety risk
- No way to fix remotely
- 1.4 million cars recalled
- August 2016: Extended to unauthorized ECU update via CAN

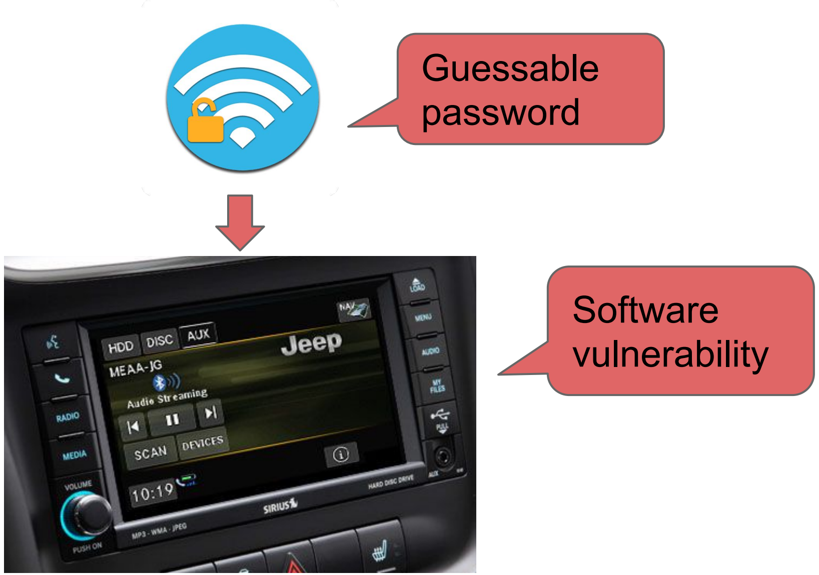
Jeep Cherokee Head Unit with Wifi

Wifi hotspot offered as a service

“Head unit”, “IVI”

- Cherokee customers can buy wifi subscription as an add-on (~$40/month)
- Connect devices in the car to the car’s wifi to get online (phones, tablets, …)
- Wifi is password protected

Wifi-based breach: Short-range

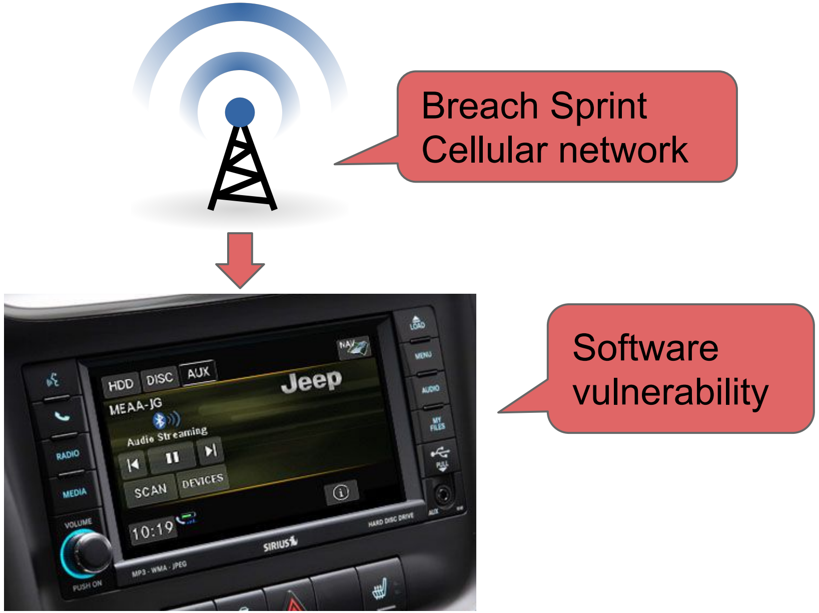


Guessable password

Software vulnerability

- Wifi **password** based on **system time after provisioning**
- January 01 2013 00:00 GMT +- 1 minute
- Multimedia system breached due to software vulnerability
- Scope: Control music player/radio/volume and track GPS coordinates when **within wifi range**

Cellular-based breach: Country-wide

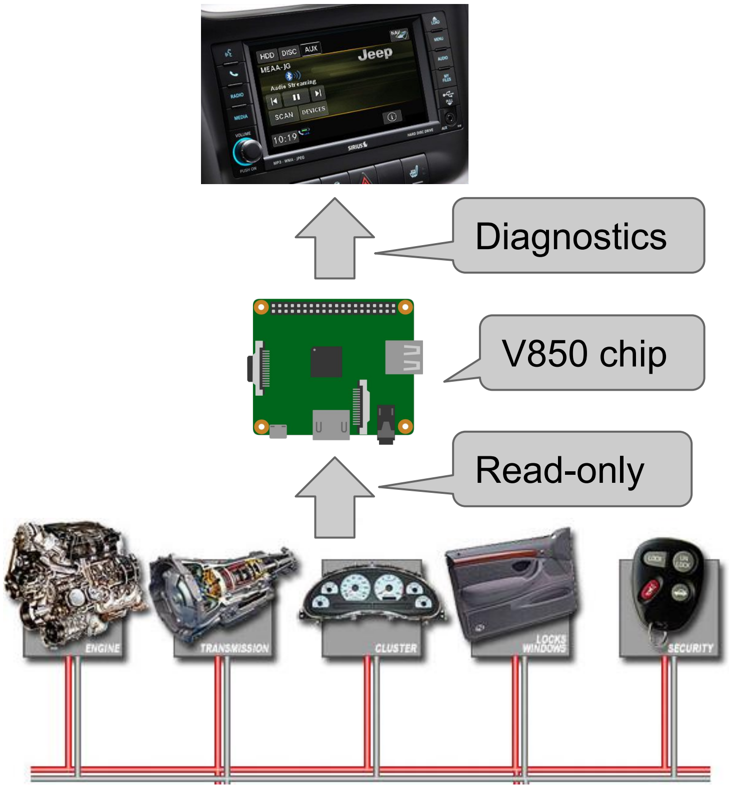


Breach Sprint Cellular network

Software vulnerability

- Scope: Control music player/radio/volume and track GPS coordinates **countrywide**
- Can also select a specific Jeep based on its GPS-coordinates

The Controller Area Network (CAN) bus


Diagnostics

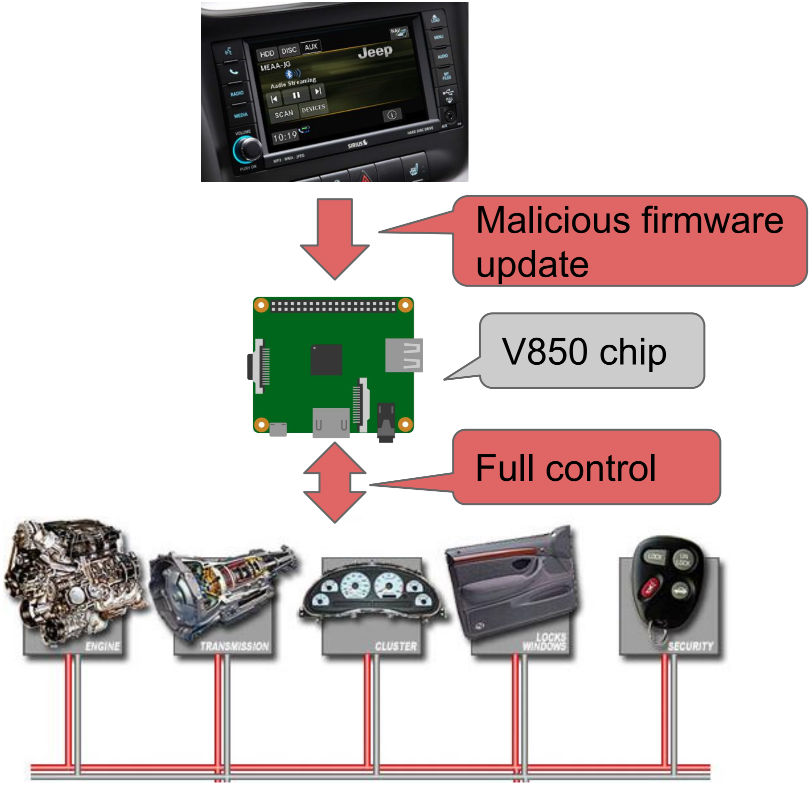
V850 chip

Read-only

- The CAN bus connects ~70 electronic control units (ECUs), including *engine control*, *transmission*, *airbags*, *braking*

- V850 chip is designed to **only read** from the CAN bus, to isolate components

CAN bus



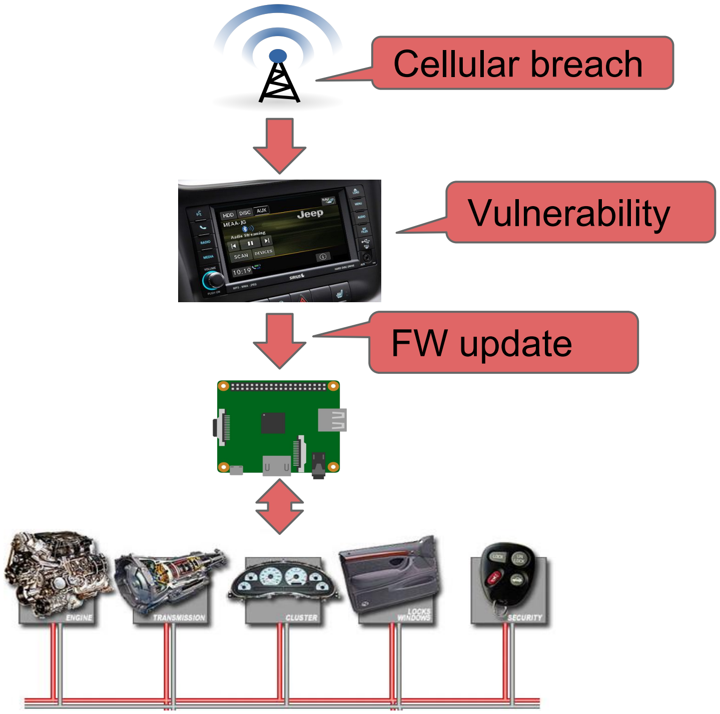
Malicious firmware update

V850 chip

Full control

- The head unit can **update the firmware** of the V850
- Firmware **update authenticity not checked** properly

Putting it together



Lessons

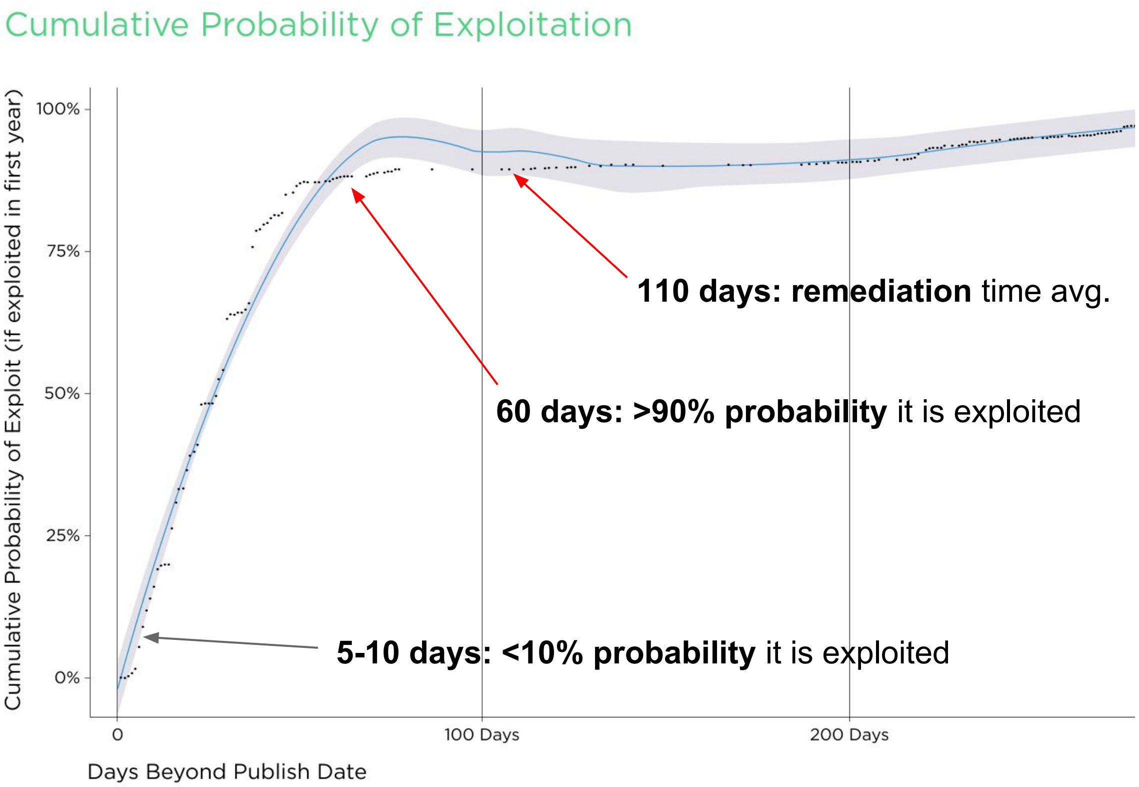
- Wifi hotspot password was predictable
- Remotely accessible service (in head unit) was vulnerable (and not updated)
- Firmware update (for V850) did not have proper authenticity checks
- The only way to fix the vulnerabilities is through a manual update (by customer or dealership)

More complexity leads to larger attack surface

- 1-25 bugs per 1000 lines of code*
  - Assume that all software components have vulnerabilities
- Rely on well-maintained software and keep it updated
  - Open source vs. proprietary is a red herring
  - Do not build all the software in-house
- Principle of least privilege
- Separation of privilege
- Kerckhoff’s principle

**Source: Steve McConnell, Code Complete*

Security patching is done too late



110 days: remediation time avg.

60 days: >90% probability it is exploited

5-10 days: <10% probability it is exploited

Source: How the Rise in Non-Targeted Attacks Has Widened the Remediation Gap, Kenna Security

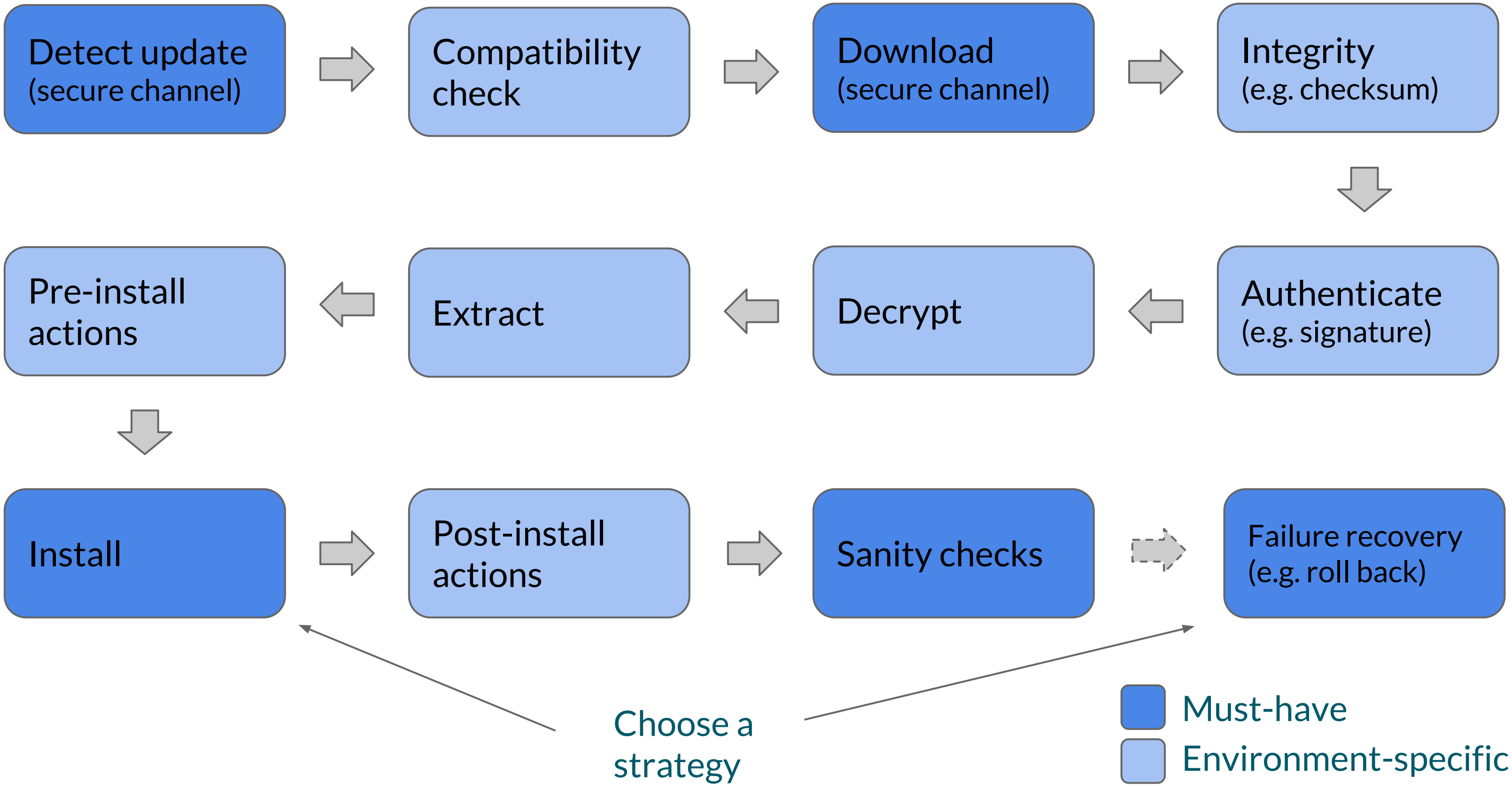
Why security patching happens too late

- The value is invisible until too late
- Too costly or risky
  - Manual? Too expensive to integrate updater?
  - Requires downtime of production? Risk of breaking production?
- Politics
- How often do *you* patch?
  - Do you have a way to do it? A process?
  - Often not a core competence and not a priority to develop updater

Patching connected devices is harder

- No/expensive physical access
  - Need failure management
- Unreliable power
  - What if power disappears in the middle of patching?
- Unreliable (wireless) network connectivity
  - Handle partial downloads
  - Ideally resume downloads in expensive networks like 3G
- Public and insecure (wireless) networks
  - Can someone inject arbitrary code during the update process?
  - Verify authenticity of update

Generic embedded updater workflow



Detect update (secure channel)

Compatibility check

Download (secure channel)

Integrity (e.g. checksum)

Authenticate (e.g. signature)

Decrypt

Extract

Pre-install actions

Install

Post-install actions

Sanity checks

Failure recovery (e.g. roll back)

Choose a strategy

🟦 Must-have

▨ Environment-specific

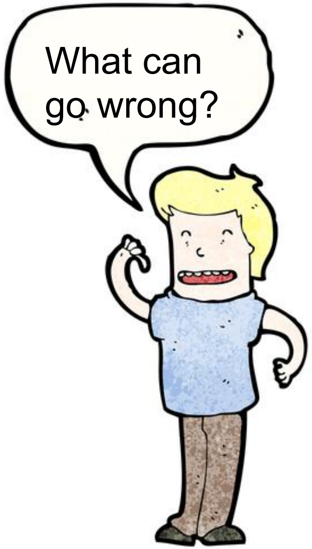
Choice of update type has tradeoffs

	Full image	Package (opkg, ...)	tar.gz	Docker/Containers
Download size	Large*	Small	Small	Medium
Installation time	Long*	Short	Short	Short
Rollback	Yes (dual partition)	Hard	Hard	Yes
Consistency	Yes	Medium	Hard	Yes
Design impact	Bootloader, Partition layout	Package manager	tar, ...	Kernel, docker

* Can mitigate with compression or binary diffs

Strategies to reduce the risk of bricking

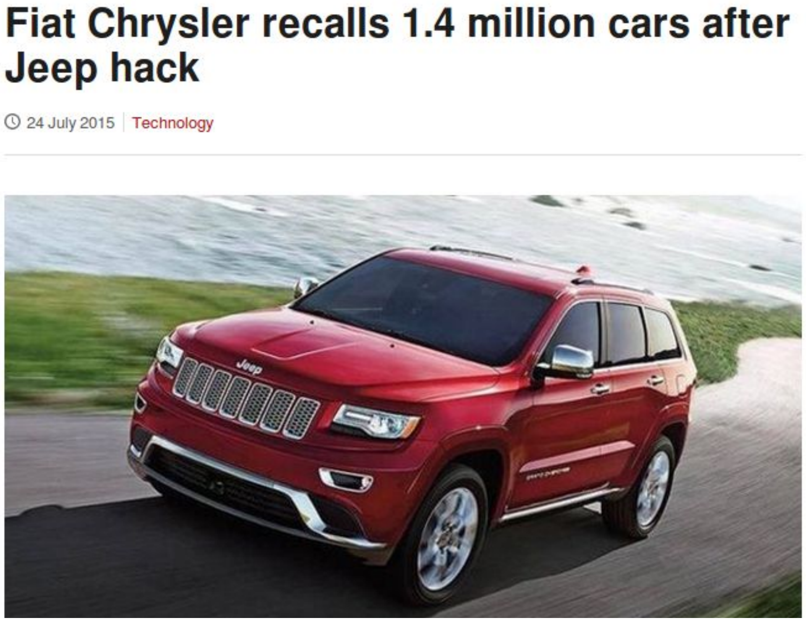
- Integrity checking
  - This must be done
  - Easy to implement
- Rollback support
  - This should be a requirement: power loss, installation error, etc.
  - Could be hard depending on update type (tarball, package)
- Phased rollout
  - I.e. don't deploy update to all devices in one go
  - Most do this to some extent: test & production environments
  - Can be more granular on device population (1%, 10%, 25%, 50%, ...)



Prepare for securing the software defined car

- Open source software where no differentiation
- Well-maintained software
- Over-the-air updates
- Apply well-known security design principles

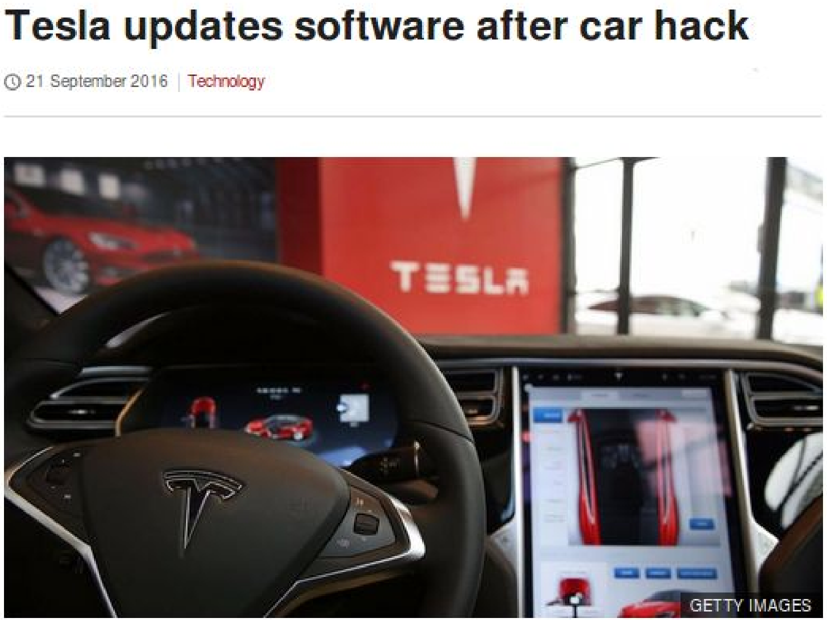
The best way to respond to hacking?



Fiat Chrysler recalls 1.4 million cars after Jeep hack

24 July 2015 | Technology

Fiat Chrysler said exploiting the flaw "**required unique and extensive technical knowledge**, prolonged physical access to a subject vehicle and extended periods of time to write code" and added **manipulating its software "constitutes criminal action"**.



Tesla updates software after car hack

21 September 2016 | Technology

Straubel [Tesla CTO] **credits KeenLabs' researchers** [...] says Tesla **will pay** KeenLabs' team a **monetary reward for its work** [...] "They did good work," Straubel says. "They **helped us find something that's a problem** we needed to fix. And that's what we did."

Sources: BBC News, Wired