Securing the Connected Car

Deploy Software Updates for Linux Devices
The software defined car

Hardware enabled
Electronics
Telematics
Infotainment
Connected
Assisted driving
1990

Software enabled
1990

Software defined
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

Differentiation

- Focus on open source here

IVI stack

- Hardware
  - Board support pkg.
  - Operating system
  - OTA updater
  - Middleware
  - Apps
  - HMI

Cost

- 10%
- 30%
- 60%

- 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

- 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

- 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

- **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

- 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.
Unauthorized update to write to the CAN bus

- 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

Cumulative Probability of Exploitation

- **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

1. Detect update (secure channel)
2. Compatibility check
3. Download (secure channel)
4. Integrity (e.g. checksum)
5. Authenticate (e.g. signature)
6. Decrypt
7. Extract
8. Install
9. Pre-install actions
10. Post-install actions
11. Sanity checks
12. Failure recovery (e.g. rollback)
13. Must-have
14. Environment-specific

Choose a strategy
## Choice of update type has tradeoffs

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

* 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 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".

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